Master's Thesis

Prediction of High Dimensional Complex Systems by Means of Generalized Local States

Sebastian Baur

Ludwig-Maximilians-Universität München
Department: Physics
Supervisor: Dr. Christoph Räth

September 15, 2020

Master's Thesis

Vorhersage hochdimensionaler komplexer Systeme mittels generalisierter lokaler Zustände

Sebastian Baur

Ludwig-Maximilians-Universität München
Fakultät: Physik
Betreuer: Dr. Christoph Räth

15. September 2020

Contents

1. Introduction

Understanding the world around us has been a human desire for as long as humans inhabited this planet. Truly understanding every minutia of the cosmos is impossible due to its unfathomable complexity. Information can only be gained by separating the important from the irrelevant. Well known techniques like principal component analysis [1, 2] and autoencoders [3] have been devised for just such a purpose.

With the same goal in mind, Pathak et al. [4] introduced local states (LS). A new approach to reduce the complexity of datasets in the context of predictions. In systems with known local interactions, this method can be used to reduce the size of a potentially very large dataset to a number of significantly smaller, much easier predictable local neighborhoods. Unfortunately, this approach is limited to systems where such a local interaction is not only present but also known a priori, severely limiting its potential applications.

This thesis introduces the method of generalized local states (GLS), lifting the restriction to locally interacting systems by abstracting spatial distances to *similarity* between time series. This similarity is derived from information driven measures and hence allows the definition of generalized local neighborhoods in systems where no spatial interactions or even spatial distance exists.

We will demonstrate this approach on the basis of reservoir computing (RC) [5], a machine learning (ML) routine with promising future, outperforming many state of the art ML methods [6, 7]. We combine this with the use of cross-correlation (CC) and mutual information (MI) as exemplary similarity measures (SMs).

As illustrative datasets we use the Lorenz-96 (L96) [8] and Kuramoto-Sivashinsky (KS) [9] systems, both well understood, locally interacting, chaotic systems. This choice allows us to compare our method of GLS to Pathak's spatial LS approach.

We want to emphasize that neither RC, nor the two illustrative systems, nor the chosen information measures are an essential part of our presented method. In principle, GLS can be used with various ML methods and similarity measures. Depending on the given dataset an appropriate choice has to be made.

In the following 2nd chapter, a summary of RC and its main differences to other ML approaches are given. Chapter 3 introduces local states and its extension to GLS. Furthermore we will explain CC and MI, the information measures used to generate the SMs, as well as the process of clustering, which allows greater scalability of the GLS. Chapter 4 covers the L96 and KS systems, the algorithms used to simulate them and the maximal Lyapunov exponent (MLE), a quantity characterizing chaotic systems. In chapter 5 we present a number of experiments comparing LS and GLS predictions, as well as quantifying GLS scalability to high dimensional systems before leading to a final discussion and outlook in chapter 6.

2. Reservoir Computing

The last decade has been a renaissance for artificial intelligence and ML. Many tasks previously thought to be almost impossible for traditional computing systems can now be solved in a very short time frame. Image classification [10, 11], time series prediction [12, 13], as well as beating the best humans in demanding games like Go [14] are all tasks that have largely been mastered by artificial neural networks (ANNs).

2.1 Reservoir Computing Introduction

Throughout the years, many different versions of ANNs have been devised with feed forward networks (FFNs) used in deep learning likely being the most prominent one. [15] While showing remarkable successes for static tasks[1], they are overshadowed by recurrent neural networks (RNNs) and their derivatives in dynamic tasks.

As with most ML systems, the core of a RNN is a network mapping an input into a higher-dimensional space which is then used to transform the data before a readout for pattern analysis or time series prediction takes place. A sketch of basic FFN and RC anatomy is depicted in 2.1.

Crucially, RNN networks exhibit topological loops which FFNs do not. RC developed as an extension of RNNs and liquid state machines shares this important feature. [5, 16] Compared to FFNs, which are often used as classifiers, the RNN loops make RC especially qualified for temporal/sequential data, as they enable the network to “remember” previous inputs.

RC distinguishes itself from the other RNN variants by keeping its internal network fixed, i.e. only the readout layer is trained. This significantly decreases the required training time, since the network can now be trained with classical methods such as linear regression. Moreover, this makes RC very amenable to physical implementations only possible due to the static nature of the reservoir. [17] Additionally, the training of the output is conventionally done with simple methods such as linear regression. Thus, irrespective of its performance, a major advantage of RC is its fast and cheap training compared to other ANNs.

[1]here defined as tasks without temporal or sequential information

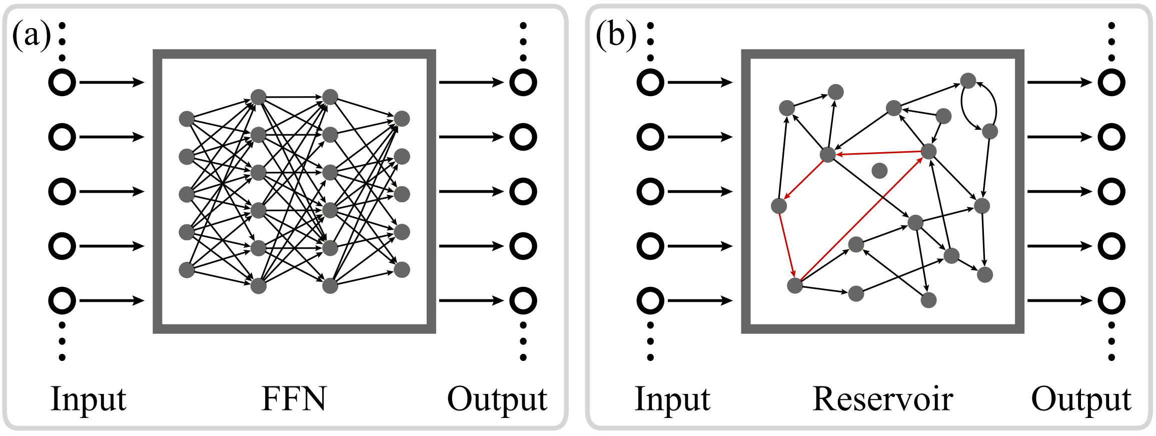

Figure 2.1: ML network schematics. **(a)** A basic FFN setup with input on the left and readout on the right. Crucially, the network in the center consists only of right facing connections, making the interaction of input data from different computation steps impossible. **(b)** A basic RC setup with the same input and output couplers as the FFN. The reservoir network in the center is fundamentally different from the FFN network as it exhibits topological loops. An example of such a loop is marked in red.

2.2 Implementation Details

The core of RC is a network, in the following called the reservoir $\mathbf{A}$. While different network topologies have been investigated in research [18, 19], sparse random $D_r \times D_r$ networks with average node degree κ and Erdős–Rényi connectivity deliver consistently great results and thus are also chosen here.

After network generation, all its connection strengths are scaled to have a predetermined fixed spectral radius ρ. Not only is the spectral radius an important hyperparameter to be optimized, it is advisable to choose $\rho < 1$ for the reservoir to gain the echo state property which is crucial during the training process [2].

The D_{in} dimensional input $\boldsymbol{x}(t)$ interacts with the reservoir state $\boldsymbol{r}(t)$ through an input coupler which in our case is a sparse $D_r \times D_{\text{in}}$ matrix $\mathbf{W}_{\text{in}}$. Following Lu et al. [20], $\mathbf{W}_{\text{in}}$ is created such that one element in each row is chosen uniformly between $[-\omega, \omega]$ where ω is the input coupler scaling parameter. All other elements of $\mathbf{W}_{\text{in}}$ are zero.

The input data then connects with the reservoir state $\boldsymbol{r}(t)$ of the previous time step via the hyperbolic tangent activation function $\tanh(\cdot)$ to advance the reservoir state by one step in time

$$\boldsymbol{r}(t + \Delta t) = \tanh\left(\mathbf{A}\boldsymbol{r}(t) + \mathbf{W}_{\text{in}}\,\boldsymbol{x}(t)\right) , \tag{2.1}$$

where $\tanh(\boldsymbol{x})$ is the vector $[\tanh(x_1), \tanh(x_2), ...]^T$. Training $\boldsymbol{r}(t)$ is recorded and used during the regression procedure as detailed below. For the activation function a wide variety

[2] $\rho < 1$ is not a necessary condition for the echo state property but a sufficient one [5]

of choices are possible. The effect of different activation functions on the prediction quality is subject of current research.

In the same way an input coupler is used to get information into the reservoir, an output coupler $\mathbf{W}_{\text{out}}$ is used to get information out of it. In the case $\mathbf{W}_{\text{out}}$ is a matrix, the training is done via a simple linear regression procedure

$$\mathbf{W}_{\text{out}} = \arg \min_{\mathbf{W}_{\text{out}}} \|\mathbf{W}_{\text{out}} \tilde{\boldsymbol{r}}(t) - \boldsymbol{y}_{\text{T}}(t)\| + \beta \, \|\mathbf{W}_{\text{out}}\| \; , \tag{2.2}$$

where $\boldsymbol{y}_{\text{T}}(t)$ is the D_{out} dimensional target output, $\|\cdot\|$ is the vector L2-norm and β is the L2 regularization (ridge regression) parameter which prevents the system from overfitting. Furthermore $\tilde{\boldsymbol{r}}$ is a nonlinear transformation of the reservoir state $\boldsymbol{r}$.

The W_{out} minimizing equation 2.2 is solved by a system of linear equations given by

$$\left(\mathbf{r}^T \mathbf{r} + \beta \mathbb{1}\right) \mathbf{W}_{\text{out}} = \mathbf{r}^T \mathbf{y}_{\text{T}} \, , \tag{2.3}$$

which, compared to many other ML training procedures, is quickly computable by modern hardware.

Note that this nonlinear transformation of the reservoir state $\boldsymbol{r}$ to $\tilde{\boldsymbol{r}}$ has proven crucial to achieve the high rate of successful long and short term predictions demonstrated in chapter 5. Here we choose $\tilde{\boldsymbol{r}} = [\boldsymbol{r}, \boldsymbol{r}^2]^T = [r_1, r_2, ..., r_{D_r}, r_1^2, r_2^2, ...r_{D_r}^2]^T$, which results in $\mathbf{W}_{\text{out}}$ being a $D_{\text{out}} \times 2D_r$ matrix. As also observed in Chattopadhyay et al. [7], a nonlinear transformed $\tilde{\boldsymbol{r}}$ is especially important for the L96 system we will work with later on.

Its flexibility allows RC in principle to be used for a variety of tasks, including classification. This is achieved by changing the target output $\boldsymbol{y}_{\text{T}}(t)$ to an appropriate quantity during the regression procedure. In this thesis we will consider the target to be a prediction of the full or partial input time series, which is the typical use case for RC.

In practice the reservoir also needs to be synchronized with the training data set before the training can begin. Without this step, the arbitrary initial state of the reservoir would influence the training and regression results. Synchronization avoids this by exploiting the aforementioned echo state property of the reservoir, which lets the system “forget" past input [5]. Therefore, each training period is preceded by synchronization steps where the reservoir state is updated sequentially as given by equation 2.1 without recording the reservoir state.

Once trained, the output $\boldsymbol{y}(t)$ can be calculated from the reservoir state $\boldsymbol{r}(t)$ as

$$\mathbf{y}(t) = \mathbf{W}_{\text{out}} \tilde{\boldsymbol{r}}(t) \, . \tag{2.4}$$

When using RC for prediction, it can be run autonomously by using the prediction of the previous time step $\boldsymbol{y}(t)$ as the input $\boldsymbol{x}_{\text{pred}}(t)$ to calculate the next predicted time step $\boldsymbol{y}(t+\Delta t)$ with. In this case one finds

$$\boldsymbol{x}_{\text{pred}}(t) = \mathbf{y}(t)\,, \tag{2.5}$$

$$\boldsymbol{r}(t+\Delta t) = \tanh\left(\mathbf{A}\boldsymbol{r}(t) + \mathbf{W}_{\text{in}}\,\boldsymbol{x}_{\text{pred}}(t)\right)\,, \tag{2.6}$$

$$\mathbf{y}(t+\Delta t) = \mathbf{W}_{\text{out}}\tilde{\boldsymbol{r}}(t+\Delta t)\,. \tag{2.7}$$

3. General Local States

Reducing the dimension of a dataset can be vital for many real world applications. By disposing of superfluous information, data processing operations not only become faster, but many methods rely on a manageable data size due to their scaling behavior. This also applies to RC as the reservoir size needed to accurately characterize and predict high dimensional input quickly becomes unfeasible. (See e.g. [4])

As such, measures and methods are needed to reduce the number of dimensions of the input data, while retaining its essential characteristics. In this chapter we will introduce the method of GLS which achieves just that.

3.1 Generalization

GLS is based on the *local state* (LS) approach published by Pathak et al. in 2018 [4]. While non-local implementations of ANN algorithms use just one network to process all input data (Figure 3.1a), LS and GLS partitions the input data into multiple subsets of smaller dimension. These subsets are in the following called *neighborhoods*.

Each neighborhood itself consists of a number of *core* dimensions and *neighbor* dimensions and is assigned its own ANN. Each ANN in question then uses only the dimensions of the input making up its neighborhood to predict its core dimensions as accurately as possible. In Figure 3.1b-c) sketches of such local neighborhoods are shown

In the context of RC this means that the input time series for the reservoir assigned to the i-th neighborhood is not the full D_{in} dimensional input time series $\boldsymbol{x}$ anymore, but instead a slice unique to this neighborhood $\boldsymbol{x}^i$ of dimension $D^i_{\text{in}} \leq D_{\text{in}}$. Similarly, the trained output of the i-th neighborhood $\boldsymbol{y}^i$ is given by just its core dimensions and hence is an even smaller subset of the neighborhood of dimension $D^i_{\text{out}} \leq D^i_{\text{in}}$.

While the training proceeds as described in chapter 2.2, the prediction needs one further adjustment. Even though in principle the choice of neighborhoods is completely arbitrary when predicting a time series, each dimension of the original input must be in one and only one neighborhood as a core dimension. This is because, as given by equations 2.5-2.7

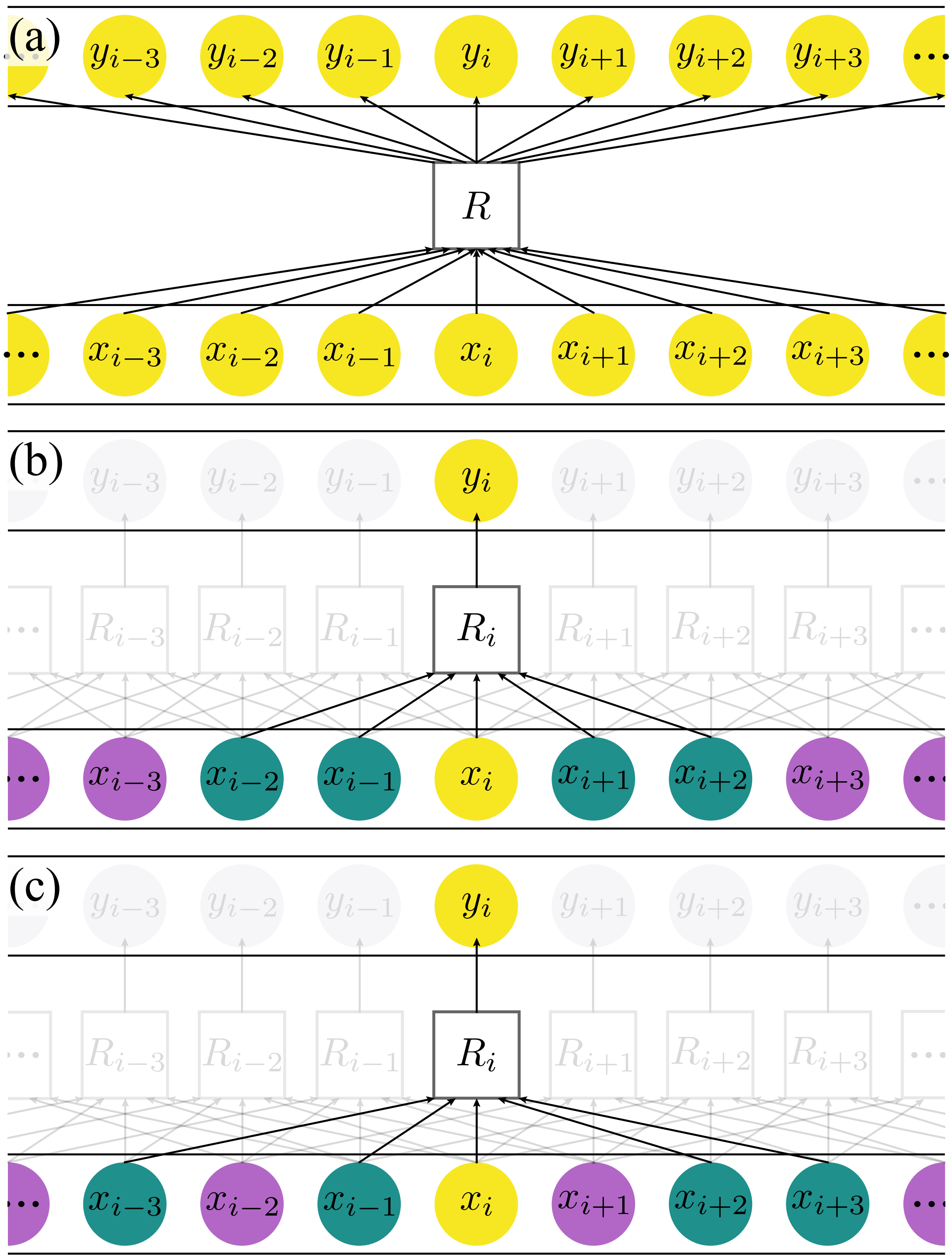

Figure 3.1: Schematic depicting different neighborhoods. Yellow circles mark the core dimensions, green the neighbors and purple all other dimensions of the full input. **(a)** The neighborhood of simple, non-local RC. All dimensions of the input vector $\boldsymbol{x}(t)$ are core dimensions. No neighbors or other reservoirs exist. **(b)** LS RC. The highlighted reservoir's neighborhood has one core dimension and four locally adjacent neighbors. Many more reservoirs exist, each with their own neighborhood. **(c)** GLS RC. As in **(b)**, the Neighborhood has one core dimension and four neighbors, with the crucial difference being that the neighbors do not have to be adjacent to the core.

between each prediction step, the neighborhoods need a new input which, as $D^i_{\text{out}} \leq D^i_{\text{in}}$ can only come from the other neighborhoods' prediction. Hence between each prediction step, a new input vector $\boldsymbol{x}_{\text{pred}}$ is formed by the combined prediction of all neighborhoods.

While having the advantage that the training of the entire system is easily parallelizable as the training of each neighborhood is completely independent of the others, its main advantage comes from the fact that it reduces the effective input dimension for each reservoir to the size of its neighborhood.

However, this procedure depends on an appropriate choice of neighborhoods, such that the information necessary to predict their core dimensions is present in the neighbors. Pathak et al. originally came up with their LS approach in the context of the KS system (see section 4.2), a purely locally interacting system. As such, their neighborhoods were chosen to have only spatially adjacent cores surrounded by contiguous buffer regions of neighbors (see Figure 3.1b).

For systems where such a local interaction is not present, this a scheme can not be used. Nonetheless, this idea of locality in the sense of importance to a prediction is generalizable as the choice of neighborhoods is practically arbitrary. For this, one needs to generalize the idea of locality to something which we will call *similarity* as well as measures to calculate it from the underlying data set.

3.2 Similarity Measures

To generate useful neighborhoods from the initial time series, we need a SM between all of its dimensions, optimally quantifying the relevance each dimension has to the prediction of each other one. For this we use two information driven measures, the CC and the MI.

3.2.1 Cross Correlation

A staple in time series analysis is the CC coefficient

$$C_{X_i,X_j} = \frac{\sum_{t=1}^{n} \left(x_{i,t} - \bar{x}_i\right)\left(x_{j,t} - \bar{x}_j\right)}{\sqrt{\sum_{t=1}^{n} \left(x_{i,t} - \bar{x}_i\right)^2}\sqrt{\sum_{t=1}^{n} \left(x_{j,t} - \bar{x}_j\right)^2}}\,, \tag{3.1}$$

where x_i is the i-th dimension of the time series $\boldsymbol{x}$ and $\bar{x}_i$ its mean.

In this normalized form the CC is bounded between -1 and 1. To transform the CC into a SM we take its absolute value

$$\text{SM}(X_i, X_j) = \left|C_{X_i,X_j}\right| \in [0, 1]\,, \tag{3.2}$$

and define a value closer to 1 as more similar. This can be justified by the intuitive reasoning that a time series doing the “opposite” of another one, contains much more relevant information for the latter’s prediction, than a completely uncorrelated one.

While straightforward to compute, the CC has the major problem that it only captures linear relationships. As chaotic time series are by their very nature nonlinear, we use one more measure that captures these nonlinear relationships, the MI.

3.2.2 Mutual Information

MI, originally defined by Shannon in 1948 [21], has a variety of properties making it ideal as a SM. In fact, “given two time series X and Y, their MI, $\widetilde{I}\left(X,Y\right) = \widetilde{I}\left(Y,X\right)$ is the average number of bits of X that can be predicted by measuring Y and vice versa" [22]. Furthermore MI is, in contrast to CC, sensitive to linear and nonlinear dependencies and becomes zero if and only if the two time series are completely independent.

In the language of probability densities the MI between two random variables X and Y is defined as

$$I\left(X_i, X_j\right) = \iint p\left(x_i, x_j\right) \log\left(\frac{p\left(x_i, x_j\right)}{p\left(x_i\right) p\left(x_j\right)}\right) dx_i dx_j \,, \tag{3.3}$$

where $p\left(x_i, x_j\right)$ is their joint probability density

While MI has many advantages over the CC, applying this definition to real, finite and often inaccurate data, can prove much trickier. Many estimators have been devised over the years, such as kernel density [23] and nearest neighbor methods [24] but, as we are working with long (1×10^5 time steps), well behaved time series we will implement the widely used binning method [21,24,25]

For this we alternatively express the MI as

$$I\left(X_i, X_j\right) = H\left(X_i\right) + H\left(X_j\right) - H\left(X_i, X_j\right) \ , \tag{3.4}$$

where $H\left(X_i\right)$ and $H\left(X_j\right)$ are the Shannon entropies of X_i and X_j respectively while $H\left(X_i, X_j\right)$ is their joint Shannon entropy. Notably this definition is only correct for discretely distributed random variables which, at least in principle, many systems are not. Both for real world data, due to a finite measurement accuracy, and simulated data, due to finite floating point accuracy, this distinction does not matter. The entropies are defined as

$$H\left(X_i\right) = -\sum_{x_i} p\left(x_i\right) \log\left(p\left(x_i\right)\right) \ , \tag{3.5}$$

$$H\left(X_i, X_j\right) = -\sum_{x_i} \sum_{x_j} p\left(x_i, x_j\right) \log\left(p\left(x_i, x_j\right)\right) \ , \tag{3.6}$$

with the expressions on the right hand side summed over all possible states of the random variables x_i .

While in principle this would be enough to calculate the MI for any discrete data set, in practice, the number of different values taken by random variables greatly eclipses the length of most time series. As such, we artificially restrict the number of possible values used in the entropy calculation by not summing over all possible states, but instead only a number of bins into which the individual values are combined. Akin to [25] we find empirically that for a time series of length T a choice of $\left\lfloor \sqrt{T/4} \right\rfloor$ bins of equal size works well for a wide range of lengths. The typical time series in this thesis has a length of $T = 10^5$ steps resulting in a total of 158 bins.

Lastly, we normalize the MI as described by Strehl et al. [26] leading to our MI SM

$$\mathrm{SM}(X_i, X_j) = \frac{I\left(X_i, X_j\right)}{\sqrt{H\left(X_i\right) H\left(X_j\right)}} \in [0, 1] \, . \tag{3.7}$$

3.3 Creating Neighborhoods

Once a SM has been chosen and calculated for all dimensions of the full input data, one needs to use it to create the neighborhoods discussed in section 3.1. As the ANN output in each neighborhood is only given by its core dimensions, predicting these core dimension as accurately as possible is most important. While other choices are possible, this leads to a simple and intuitive way of defining neighborhoods when there is only one core dimension.

3.3.1 Single Core Neighborhoods

In the LS case of a single core neighborhood, a spatial neighborhood (SN), the core's neighbors are simply the dimensions spatially closest to that core. In the GLS case with one core, the exact analog is possible, where the one core's neighbors are the dimensions most similar to it as defined be the SM. We call the corresponding neighborhoods CC or MI neighborhoods respectively, depending on the similarity measure used. All single core neighborhoods discussed during this thesis were created this way.

3.3.2 Multi Core Neighborhoods

Generating SNs with more than one core is done as in [4] by dividing the input dimensions D into g non-overlapping groups of equal size. To this group of core dimensions are then a number of neighbors added in ascending order of spatial distance to the average core in group

g. Once the desired number of neighbors is added, this group is the neighborhood.

For GLS neighborhoods with more than one core dimension, the generalization process is a little less obvious. As cores do not necessarily have to be adjacent dimensions.

Due to possibly arbitrary distance measures defined by the SM, we need to use a clustering method that can work with such arbitrary distances. Many standard clustering methods which rely on a distance metric defined between all possible points of a coordinate space, like K-Means [27], can therefore not be used here.

A popular algorithm that can work with such constraints is called agglomerative clustering [28]. In the language of dimensions and neighbors, its algorithmic implementation can be summarized as follows:

1. Initially assign each input dimension to its own neighborhood $N_1, N_2, ..., N_D$
2. Merge neighborhoods until the desired number of neighborhoods is reached
   (a) Find the closest pair of neighborhoods (N_i, N_j) according to the linkage distance $d(N_i, N_j)$.
   (b) Merge the two neighborhoods.

While many different choices for the linkage distance are possible, we use average linkage during this thesis as we heuristically found it to give the best results. It is defined as

$$d\left(N_i, N_j\right) = \frac{1}{|N_i| \cdot |N_j|} \sum_{\mathbf{x}_i \in N_i, \mathbf{x}_j \in N_j} \mathrm{SM}(x_i, x_j)\,. \tag{3.8}$$

4. Systems and Simulations

For modeling high dimensional, spatiotemporal, chaotic systems, the Lorenz-96 and Kuramoto–Sivashinsky systems have become widely used in the RC community [4, 6, 29, 30]. As such, they are the perfect example systems allowing us to compare and quantify the LS and GLS approaches in well understood systems.

4.1 Lorenz-96

Our first artificial system is the Lorenz-96 system, originally developed and often used to study climate and weather predictability [8, 31]. It is defined as

$$\frac{d\boldsymbol{x}_j}{dt} = \left(\boldsymbol{x}_{j+1} - \boldsymbol{x}_{j-2}\right)\boldsymbol{x}_{j-1} - \boldsymbol{x}_j + \boldsymbol{F}\,, \tag{4.1}$$

where $\boldsymbol{x}(t)$ is the system's D-dimensional state vector and $\boldsymbol{F}$ the forcing parameter.

During this thesis we will be looking exclusively at a forcing parameter of $|\boldsymbol{F}| = F = 5$. For this parameter choice, the L96 system exhibits both periodic and chaotic behavior, depending on its system size D as demonstrated by Marwan et al. [32]. We simulate the L96 system using the fourth order Runge-Kutta Method [33] with a time step size of $\Delta t = 0.05$. Two examples of the L96 behavior are shown in Figure 4.1.

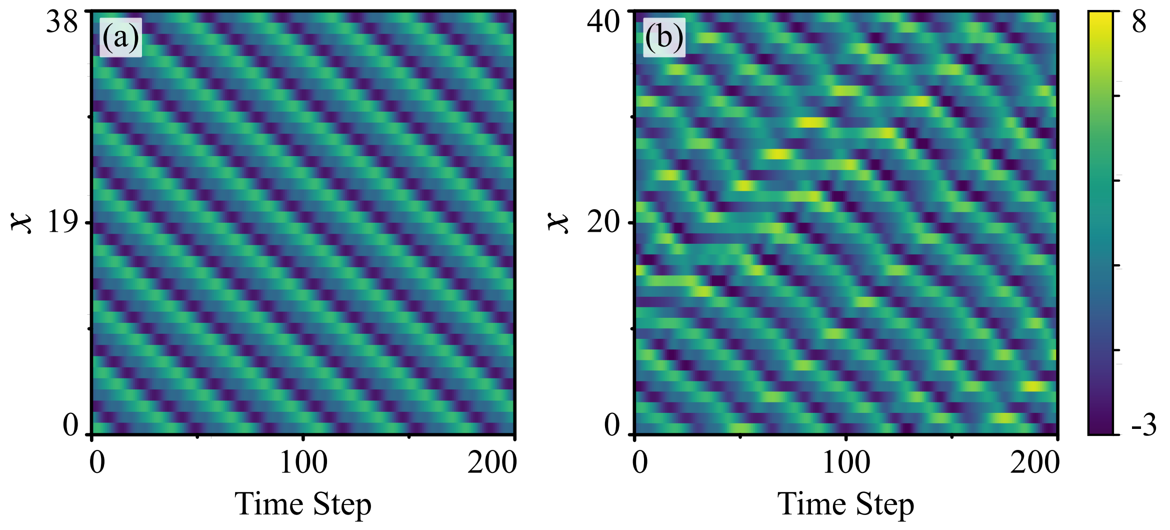

Figure 4.1: Space–time plots of the L96 system. **(a)** $D = 38, F = 5$, periodic dynamics, **(b)** $D = 40, F = 5$, chaotic dynamics.

4.2 Kuramoto-Sivashinsky

The second model, widely used to model a variety of weakly turbulent fluid systems [34, 35], is the KS system [9]. Its PDE reads

$$\partial_t u + \partial_x^4 u + \partial_x^2 u + u \partial_x u = 0 \,, \tag{4.2}$$

where the field $u(x, t)$ is defined on some domain size L. As with the L96 system, the KS system is exhibits both periodic and chaotic behavior depending on parameter choice, an example of which is depicted in Figure 4.2. In this thesis, we constrain ourselves to a domain size of $L = 22$ with periodic boundary conditions

$$u(x + L, t) = u(x, t) \qquad \text{for all } 0 \leq x \leq L \,. \tag{4.3}$$

As proven in section 4.3, this parameter choice also results in a chaotic system.

For the numerical treatment, the equations are discretized on a grid of $D = 40$ points, the same size as the Lorenz-96 system, and numerically integrated with a step size of $\Delta t = 0.5$, using the fourth order time-stepping method exponential time-differencing fourth-order Runge-Kutta (ETDRK4) [36, 37].

While widely used to simulating the KS system [6, 37–40] ETDRK4 is inherently numerically unstable [37]. As such, a well chosen initial condition is paramount for reproducing a sensible realization of the KS system. Hence, unless otherwise specified, all simulations of the KS system are initialized by a generalized form of the initial condition used by Kassam et al. [37]. This initial condition is given together with our python code for the KS simulation and a

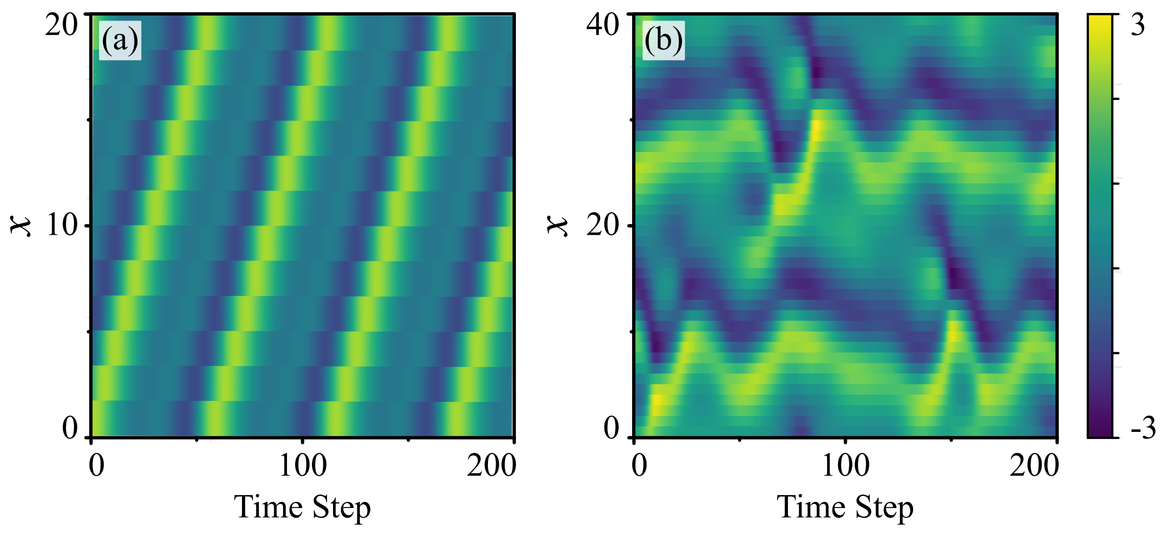

Figure 4.2: Space–time plots of the KS system. **(a)** $L = 12$, $N = 20$, periodic dynamics, **(b)** $L = 33$, $N = 40$, chaotic dynamics.

comparison of the initial system dynamics in appendix A.1.

4.3 Lyapunov Exponents

Chaotic systems can be characterized by a variety of quantities, chief among them being the spectrum of Lyapunov exponents, "which has proven to be the most useful dynamical diagnostic for chaotic systems" [41].

Typically considered the most important Lyapunov exponent is the MLE defined as the largest Lyapunov exponent of any given chaotic system. Its importance stems from the fact that it is intimately tied to the predictability of the system as, given a MLE Λ_{max} two infinitesimally close trajectories in phase space, initially separated by the vector $\delta\boldsymbol{x}(t=0)$ diverge as [42]

$$|\delta\boldsymbol{x}(t)| \approx e^{\Lambda_{\text{max}} t} \, |\delta\boldsymbol{x}(t=0)| \, , \tag{4.4}$$

where t is the time since separation.

Due to its importance, the MLE is an ideal candidate to quantify the long term statistical properties of a chaotic system. As we will compare simulated with predicted time series in chapter 5 we seek a method which is able to calculate the MLE of both.

Many popular methods for low dimensional dynamic systems can not be used due to the high dimensionality of the GLS RC system. Popular QR decomposition based methods for example have an algorithmic complexity of $O(n^3)$ where n is the size of the square Q and R matrices and as such the dimension of the dynamic system. [43] [1]

While this problem can be somewhat alleviated by rounding techniques, GPU computing and parallelization on a computing cluster, its scaling behavior poses a clear obstacle for the generalization to higher dimensions which is one of the main goals of GLS. As such, we limit ourselves to compute only the most important, largest Lyapunov exponent and take advantage of the fact that we can repeat our predictions and simulations as often as we want, which allows us to use Sprott's method of orbit separation (OS) [44].

By taking the logarithm of equation 4.4 we obtain

$$\log|\delta\boldsymbol{x}(t)| \approx \log|\delta\boldsymbol{x}(t=0)| + \Lambda_{\text{max}} t \, , \tag{4.5}$$

which is a linear relationship between the two measurable quantities $\log|\delta\boldsymbol{x}(t)|$ and t. Taking the average $\langle\cdot\rangle$ over many trajectory divergences in different parts of the chaotic attractor we find

$$\Lambda_{\text{max}} = \frac{1}{t_2 - t_1} \left\langle \frac{\log|\delta\boldsymbol{x}(t_2)|}{\log|\delta\boldsymbol{x}(t_1)|} \right\rangle \, . \tag{4.6}$$

[1]With a typical time of computing the Lyapunov exponent for a 500 dimensional system of about one second, a naive application of the algorithm for the 40×5000 dimensional GLS RC used in chapter 5 would take about 2 years and, assuming 8 byte floating point values, consume a minimum of 320GB of memory.

Note that for this equation to hold, we are only using the divergence data after transient effects have subsided but before the divergence size saturates due to it reaching the size of the chaotic attractor.

From this, we can use a linear least squares fit to calculate the MLE. Note that this averaging is essential to getting useful results,especially for noisy data [42].

Additionally, we can use the MLE to calculate the Lyapunov time T_L, i.e., the average amount of time for errors to grow by a factor of e as

$$T_L = \frac{1}{\Lambda_{\max}} \,. \tag{4.7}$$

5. Experiments

In the following set of experiments we explore and compare the LS and GLS methods. We will analyze the neighborhoods created by the two different approaches as well as the resulting short and long term RC predictions using these neighborhoods.

5.1 System Characterization

First and foremost, we calculate the MLE of the L96 and KS systems, as we will use it later to quantify the accuracy of our long term RC predictions.

We do so by using OS as described in section 4.3. Both systems are first simulated using the parameters and methods described in chapter 4 for 1×10^5 time steps after discarding the first 2×10^4 time steps to avoid the influence of transient effects on the MLE fit. These 1×10^5 time steps are then used as basis for the OS. From these 1×10^5 trajectory positions, we take every tenth one, leaving us with 1×10^4 and add to them normally distributed noise with standard deviation $\sigma_{\text{noise}} = 1 \times 10^{-10}\sigma_{\text{data}}$, with σ_{data} being the standard deviation of the simulated data. Using the noisy trajectory positions as a new starting point, we simulate the system for 1500 time steps from each one and compute the separation magnitude from the initial, noise-less time series.

Lastly, we use a linear least squares fit as described in section 4.3, resulting in Figure 5.1 with the relevant quantities listed in table 5.1

System	Λ_{max}	Δt	T_L,	time steps per T_L,
KS	0.049	0.5	20.4	40.8
L96	0.45	0.05	2.2	45

Table 5.1: MLEs Λ_{max}, time step size Δt, Lyapunov time T_L, and the number of time steps per T_L for the KS and L96 systems.

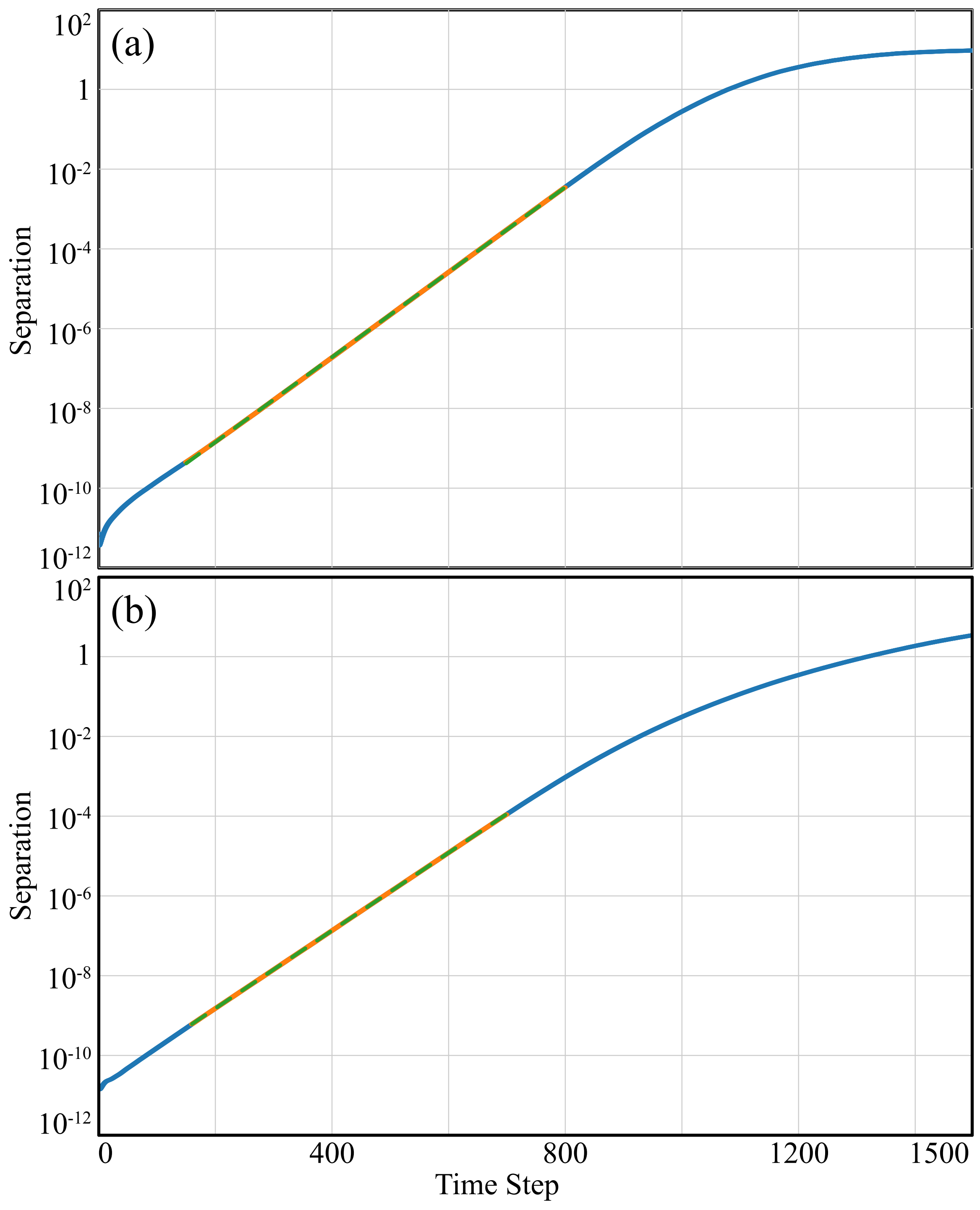

Figure 5.1: OS for 1×10^4 diverging trajectories in the **(a)** KS and **(b)** L96 system. The measured data is depicted in blue, the subset used for the linear regression fit in orange, while the striped green line depicts the resulting linear fit.

5.2 Parameters

To enable a fair comparison of the different neighborhood generation methods, the RC hyperparameters are optimized for the LS neighborhoods and then copied for the GLS neighborhoods without further adjustments. This slightly unorthodox approach is justified because we will compare the prediction efficacies of GLS and LS. As RC is sensitive on its hyperparameters [19], our approach of equal hyperparameters makes such a comparison much more straight forward.

Furthermore we optimize the hyperparameters for the LS and copy them to the GLS, and not the other way around, such that that any difference in efficacy resulting from the hyperparameters can only benefit the LS, the method not introduced in this thesis.

Additionally, this was done with the goal in mind to predict a mixed L96-KS, non-locally interacting system in section 5.4. As we will see later, the SN approach fails at this task. Nonetheless, we want to give the SN approach the highest possible chance of succeeding in this task, even though optimizing its hyperparameters on the mixed L96-KS directly is fruitless. Therefore we optimize it on the the individual L96 and KS systems simultaneously.

This optimization was done via trial and error for the short term prediction accuracy of both systems at once, with the goal of finding a hyperparameter set able to learn and predict both systems in isolation. One hyperparameter not optimized with this approach is the noise we added to all training data following Vlachas et al. [45]. Without this noise we found the short term prediction accuracy to often be higher, but at the cost of an increased rate of failed realizations[1], and lower quality long term predictions in general. This noise proved decisive in minimizing variance between network realizations and reducing the number of failed realizations, especially for the the L96 system.

Heuristically, we found normally distributed noise with standard deviation $\sigma_{\text{noise}} = 1\% \, \sigma_{\text{data}}$ where σ_{data} is the standard deviation of the training data, to be a sweet spot.

The hyperparameters used for all RC in this chapter are given in table 5.2

reservoir dimension	D_r	5000
average node degree	κ	3
spectral radius	ρ	0.5
input coupler scaling	ω	0.5
ridge regression parameter	β	10^{-6}
noise level	α	1%

Table 5.2: RC hyperparameters used during chapter 5

[1]producing only diverging predictions

The data used in the following sections is simulated as detailed in chapter 4. Transient effects of the simulated data were discarded before any synchronization, training or prediction took place. Notably, the first 2×10^4 time steps after the start of a simulation are dropped to make sure the starting condition of the simulation does not influence the results. Similarly, each reservoir training and prediction is preceded by 2000 synchronization steps.

Each reservoir network is randomly generated as described in section 2.2. No randomly generated network was discarded due to unsatisfactory prediction accuracy or any other reasons.

All reservoirs were trained for a total of 1×10^5 time steps using the noisy training data, which is a typical training data length for higher dimensional systems [4, 7].

5.3 Locally Interacting Systems

In this section the predictive efficacy of GLS and LS neighborhoods for the locally interacting L96 and KS systems are compared.

5.3.1 Neighborhoods

In the realistic case of having only a finite amount of measured data, the training data is often the only data one has to characterize a system. As such, we restrict ourselves to use only this training data to calculate the neighborhoods. Additionally, as measured time series would also be noisy in reality, we do not remove the noise added to the training data in section 5.2 when calculating the SMs of the CC and MI neighborhoods. A side effect of this is that the neighborhoods for each random network realization will look slightly different due to this noise. Therefore all neighborhoods shown in this thesis are representative examples, but not uniform for all realizations.

The neighborhoods are calculated as described in section 3.3.1 for a single core and 18 neighbors, in the case of SN and MI neighborhoods, and 28 neighbors, in the case of a CC neighborhood. Example neighborhoods for the KS and L96 systems are shown in Figure 5.2 and Figure 5.3 respectively. While the SN neighborhoods are defined as a contiguous set of cores (here just one) and their nearest neighbors, the CC and MI neighborhoods warrant of a closer look. First and foremost, even though the MI neighborhoods were calculated dynamically, without directly using the knowledge of KS or L96 being a locally interacting system, the resulting neighborhoods closely resemble the SN neighborhoods. The CC neighborhoods in contrast include many dimensions spatially far away from the core, especially for the L96 system.

This tendency of dimensions spatially distant to the core being included in the CC neighborhood

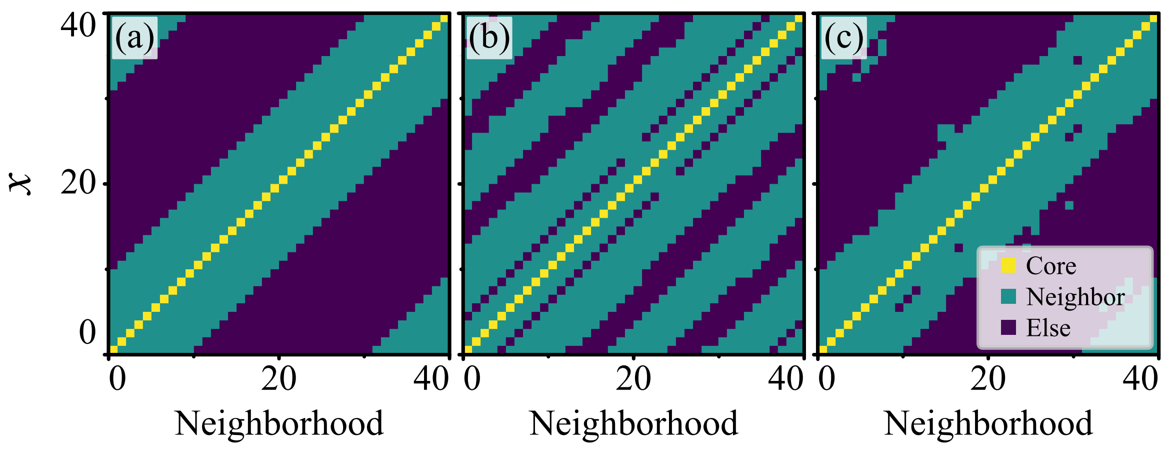

Figure 5.2: Example Neighborhoods for the KS system. Each row depicts one dimension of the time series, while each column represents a neighborhood. Each neighborhood is defined by the core dimension (yellow) and its neighbors (green). **(a)** SN neighborhoods. As defined in section 3.1 the SN neighborhood consists only of a core and its 18 nearest neighbors. **(b)** CC neighborhoods. As mentioned in the text, each CC neighborhood consists of a total of 29 dimensions. **(c)** MI neighborhoods.

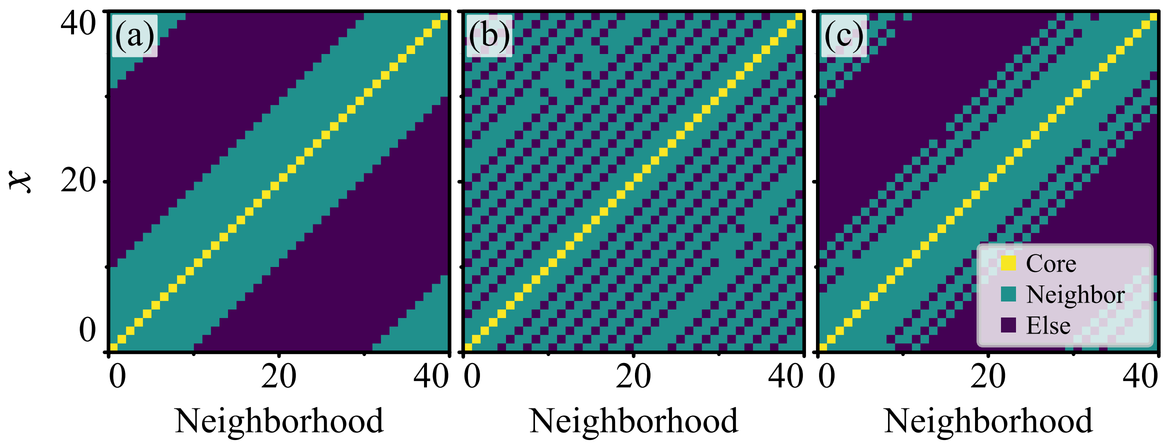

Figure 5.3: Example Neighborhoods for the L96 system. Each row depicts one dimension of the time series, while each column represents a neighborhood. Each neighborhood is defined by the core dimension (yellow) and its neighbors (green). **(a)** SN neighborhoods. As defined in section 3.1 the SN neighborhood consists only of a core and its 18 nearest neighbors. **(b)** CC neighborhoods. As mentioned in the text, each CC neighborhood consists of a total of 29 dimensions. **(c)** MI neighborhoods.

is also the reason that CC neighborhoods are larger than the MI and SN ones. While the neighborhood sizes for SN and MI were chosen to be similar to Pathak's original paper [4] which had a total neighborhood size of 20[2]. While for them this results in good predictions, for the CC SM this leads to essentially all predictions diverging. This is likely the result of the CC SM not “recognizing" the importance of the core's nearest neighbors in these systems. As such, the CC neighborhood size was increased to a total size of 29, the minimum where no predictions diverged and, not coincidentally, where the core's nearest neighbors were consistently included in its neighborhood. CC neighborhoods of size 19, 27 and 29 for the L96 system are depicted in Figure 5.4. While the real importance of each dimension regarding

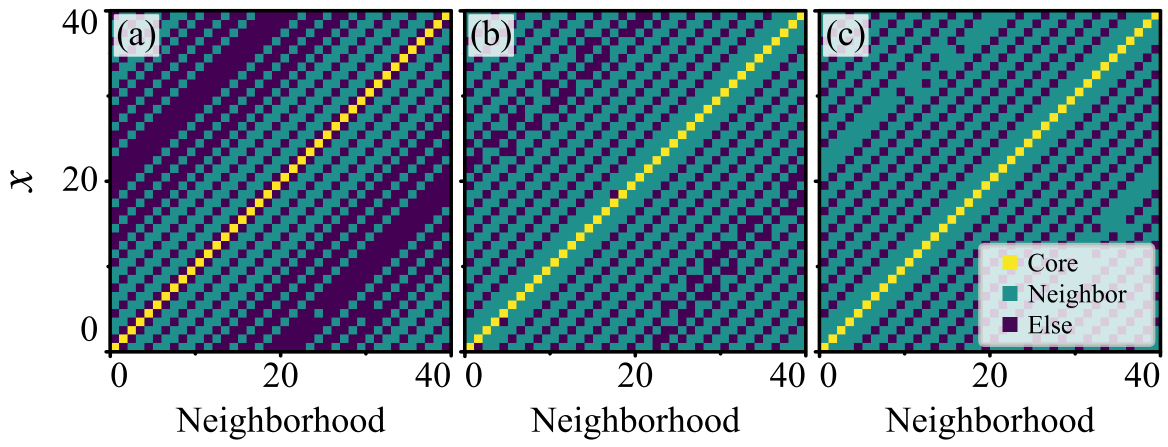

Figure 5.4: CC neighborhoods of the L96 system. **(a)** Total neighborhood size 19, the core dimensions' nearest neighbors are not included in the neighborhood. **(b)** Neighborhood size 27. Only a couple nearest neighbors of the core's are missing. Nevertheless, this is fatal when trying to predict the system using these neighborhoods. **(c)** Neighborhood size 29. All core's nearest neighbors are included in the neighborhood.

the prediction of another is of course unknown, this is the problem the SM were defined to solved for after all, at least for the locally interacting L96 and KS systems studied here, the core's spatial nearest neighbors are absolutely crucial for achieving a sensible prediction.

5.3.2 Short Term Predictions

Following section 2.2, 180 distinct random network realizations are generated, 30 for each system-SM combination, using the hyperparameters of section 5.2. They are then trained as described in section 5.2 and used to predict the same 300 sections of 10 Lyapunov times (see table 5.1) length on the chaotic attractor of the L96 and KS systems. Before each prediction, the reservoirs are synchronized for 2000 steps.

To quantify the short term prediction accuracy, we define the normalized root mean square

[2]albeit with 8 core dimensions per neighborhood

error (NRMSE) as

$$\mathrm{NRMSE}(\hat{\boldsymbol{y}}) = \frac{\sqrt{\left\langle (\hat{\boldsymbol{y}} - \boldsymbol{y})^2 \right\rangle}}{y_{\max} - y_{\min}}, \tag{5.1}$$

where $\hat{\boldsymbol{y}} \in \mathbb{R}^D$ is the prediction at a single time-step, $\boldsymbol{y} \in \mathbb{R}^D$ the true signal and $y_{\max}$ ($y_{\min}$) the largest (smallest) value taken of any dimension in the simulated data set.

Short term prediction results are shown in Figures 5.5 and 5.6.

Looking at the short term predictions of the KS system, it is very striking that the averaged NRMSE of all three neighborhoods coincide more or less exactly. While the same holds true for the SN and MI neighborhoods of the L96 system, the CC prediction is significantly worse. This performance drop is likely the result of the CC neighborhood's inclusion of many dimensions which are far from the, at least for the locally interacting systems, ostensibly most important, close region around the core (see Figure 5.3b).

Comparing these results with the short term predictions of similarly sized KS and L96 systems reveals that we achieve slightly worse but nonetheless comparable performance for both. [4, 6] Considering that our hyperparameters were optimized for two different systems at the same time, a slight performance drop in the short term prediction accuracy was to be expected.

It should be noted that the method stability shown here is remarkable, as often RC algorithms have a much larger spread of prediction qualities between different random networks [19]. This stability is almost certainly attributable partly to finding the correct hyperparameters for the systems at hand, as suboptimally chosen hyperparameters often leave the best predictions intact while increasing the variance towards bad/completely failed predictions drastically. [6, 19] However, the role of the noise added to the training data in achieving this stability, especially for the L96 system, can not be understated either. Additionally it should be noted that, as is typical for RC, the variance in short term NRMSE between different starting positions on the attractor is much larger than the variance for different random network realizations. This is demonstrated in appendix A.2.

5.3.3 Climate Reproduction

Not only short term prediction accuracy is important when it comes to judging the quality of a reservoir's understanding of the underlying true data. The long term statistical, often called climate, properties of a system are, depending on the application, equally if not more important.

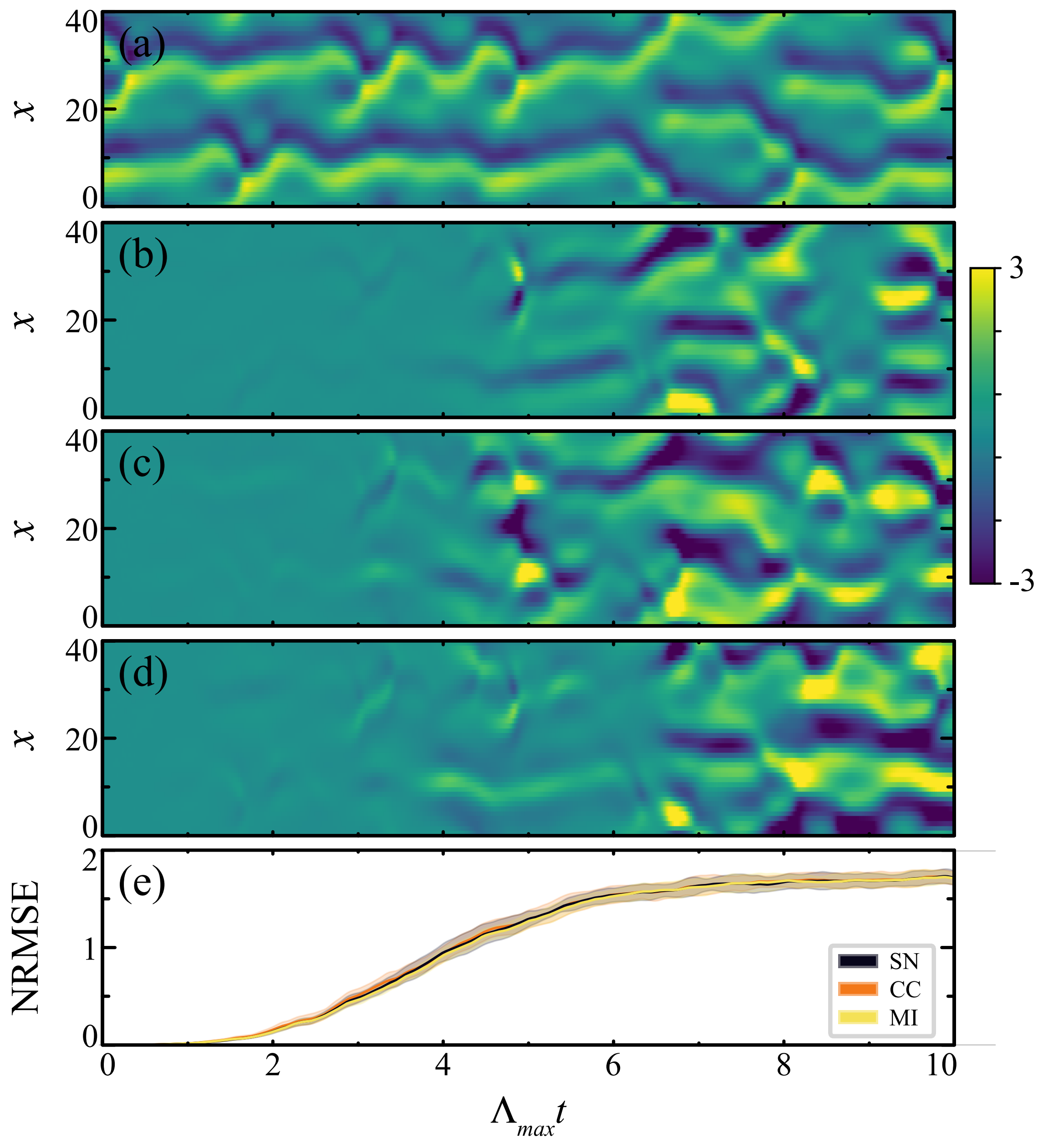

Figure 5.5: Short term prediction of the KS system. **(a)** Actual KS simulation data. **(b-d)** Exemplary error in the RC prediction when using the **(b)** SN, **(c)** CC and **(d)** MI neighborhoods. **(e)** NRMSE of SN, CC, and MI prediction data averaged over first the 300 predicted sections and then the 30 network realizations. The error bands correspond to the 3σ standard deviation of the random network realizations. We multiply t by the MLE Λ_{max} of the model, so that each unit on the horizontal axis represents one Lyapunov time.

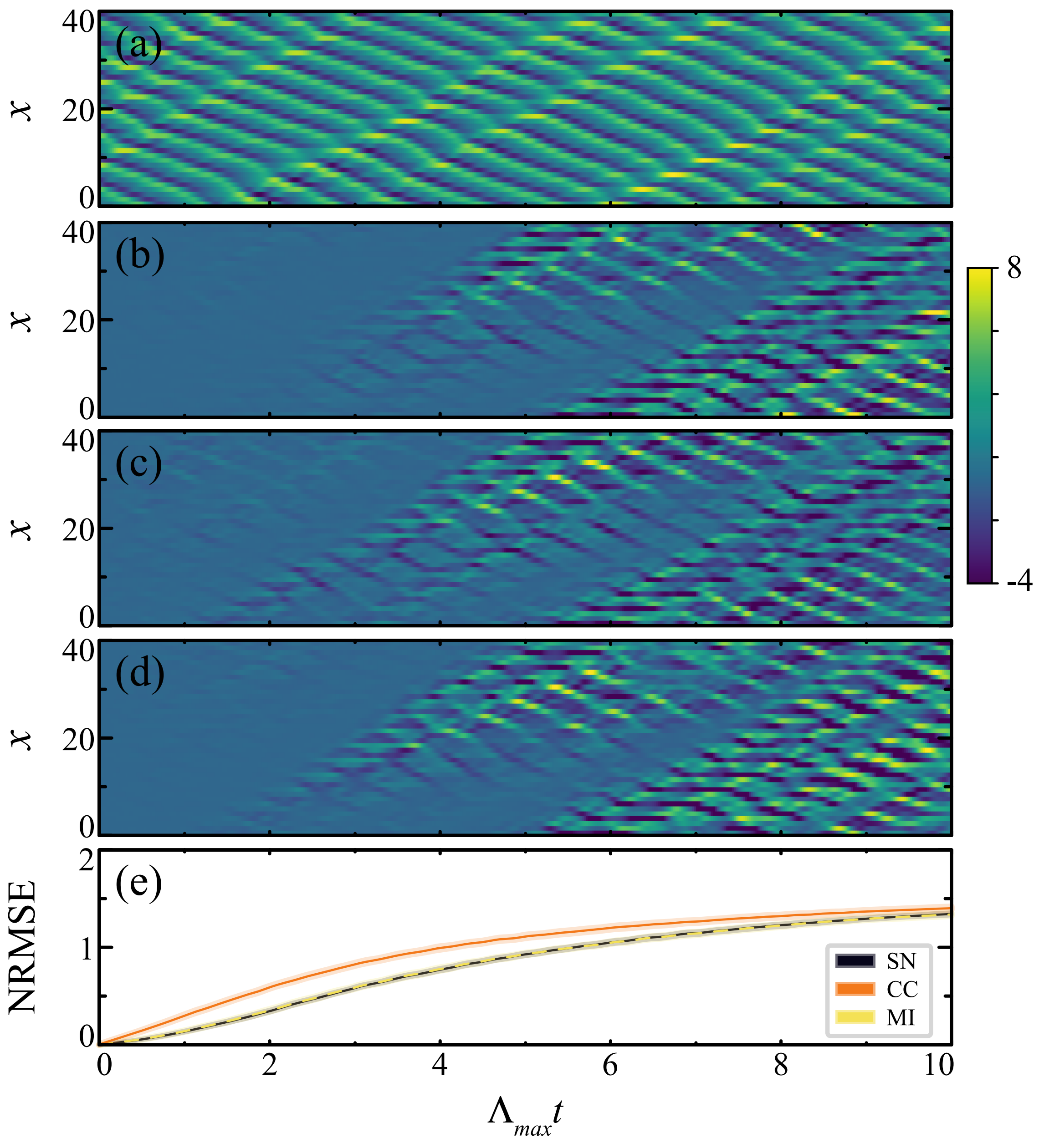

Figure 5.6: Short term prediction of the L96 system. **(a)** Actual L96 simulation data. **(b-d)** Exemplary error in the RC prediction when using the **(b)** SN, **(c)** CC and **(d)** MI neighborhoods. **(e)** NRMSE of SN, CC, and MI prediction data averaged over first the 300 predicted sections and then the 30 network realizations. The error bands correspond to the 3σ standard deviation of the random network realizations.

Probability Distribution Functions

One measure of a systems climate is the probability distribution function (PDF), which can be estimated via a simple histogram. For this purpose we let each of our reservoirs predict three distinct sections of the systems attractor each 1000 Lyapunov times in length. For the purpose of PDF estimation the three resulting predictions and reference data sets from the simulation are treated as one single data set of 3000 Lyapunov times length. The resulting 40 dimensional datasets are flattened, creating two one dimensional time series of $40 \times 3000 \times 3 = 1.2 \times 10^5$ Lyapunov time length which are used to build the histogram. The histograms themselves consist of 100 equally sized bins. To make comparison easy, they are chosen such that bin position and size for the simulated and predicted time series histograms are the same.

Looking at the PDF calculated for the KS system in Figure 5.7 it is clear that the CC neighborhood is able to reproduce the underlying simulated data the best, with excellent agreement even at the tails of the distribution. Next best is the MI which, while still showing close agreement at the core of the distribution, exhibits a divergence at the tails mainly by assuming values of greater magnitude than are ever reached in the simulated dataset. This pattern continues with the SN which shows the same divergence behavior as the MI RC, only at a more prominently.

The explanation for this is that both the MI RC as well as the SN RC seem to "get stuck" in different parts of the attractor when predicting independently for a long time. Crucially, this is not a full breakdown of the trajectory dynamics as can be seen when looking at exemplary trajectory slices, individual realization PDFs and the longterm NRMSE depicted in Figure 5.8 Looking at the predicted data more closely, one finds the characteristic shapes and length scales of the KS still intact, but with the normally almost horizontal lines showing a clear drift to lower or higher dimensions respectively. While this behavior is also present in the simulated KS data, this is the case only for much shorter durations of at most a couple Lyapunov times, while the predicted data manifests this shift for hundreds of Lyapunov times or more. Notably no clear separation between the start and end of such a the shift period is visible in the NRMSEs even though the effect on the PDFs is immediately noticeable.

This is especially interesting as the SN and MI neighborhoods for the KS system are almost the same (see Figure 5.2), meaning that the resulting difference in climate reproduction ability must come from the few outliers, far removed from the ostensibly most important closest neighbors. These results indicate that even for purely locally interacting systems there might be something to gain from using the GLS approach over the conventional LS one, or at least further reinforces the importance of small amounts of randomness to get high quality long term predictions from RC.

The L96 PDFs presented in Figure 5.9 on the other hand are up to and including the distribution's tails in excellent agreement with the simulated data for all three neighborhood

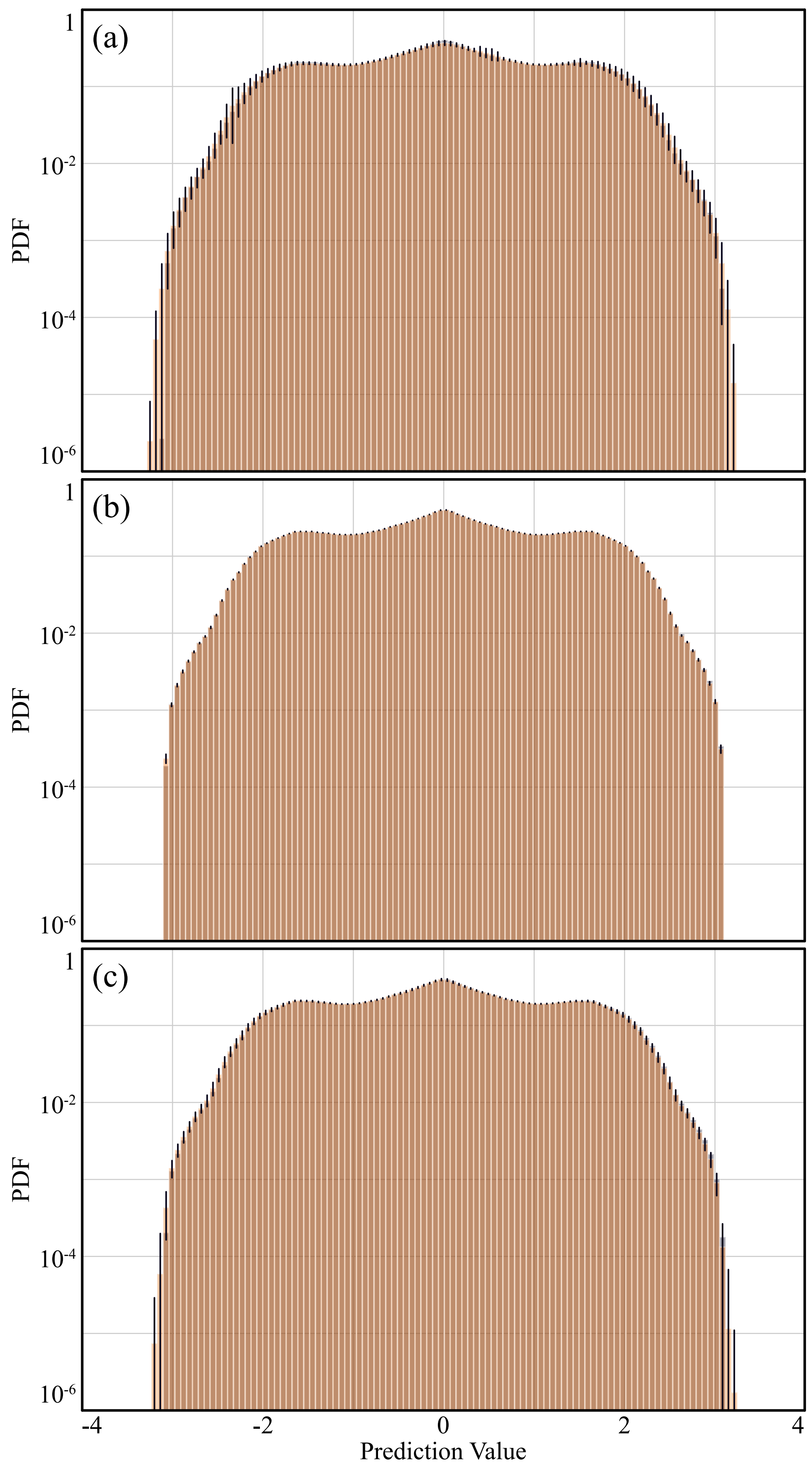

Figure 5.7: PDF for the longterm predictions of the KS system. The PDFs are estimated via two superimposed 100 bin histograms of the simulated (gray) and the predicted (orange) data of the **(a)** SN, **(b)** CC and **(c)** MI Neighborhoods. Each entry in the histogram is generated by the prediction/simulation of a single dimension value, with the dataset coming from three 1000 Lyapunov time long sequences. The errorbars represent the 1σ standard deviation of the predicted histogram bins.

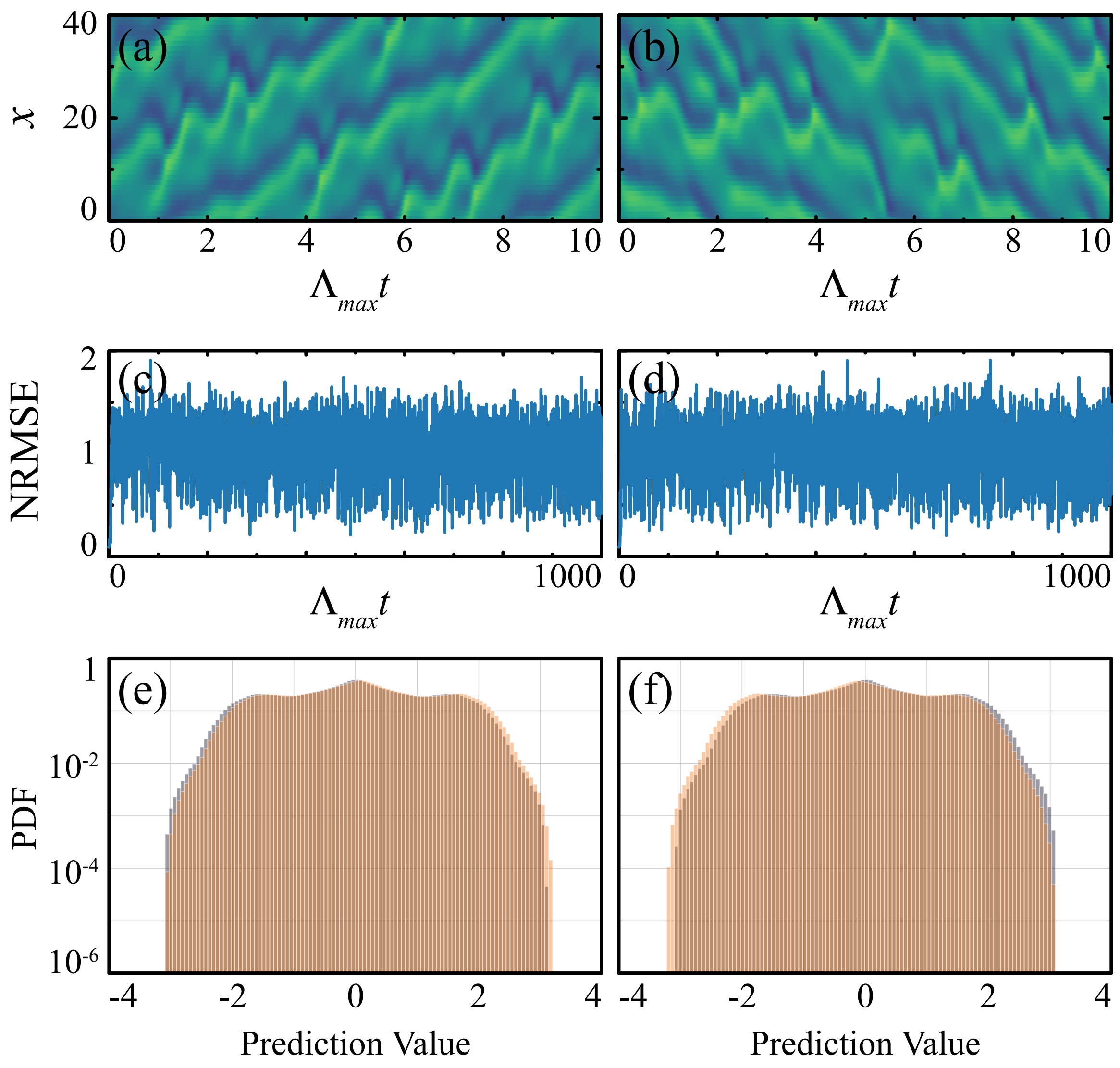

Figure 5.8: SN example realizations exhibiting the PDF shift. **(a)-(b)** Predictions of the KS system, qualitatively different than the simulated data. **(c)-(d)** NRMSEs over a full 1000 Lyapunov times corresponding to the predictions shown in (a) and (b) respectively. **(e)-(f)** PDFs corresponding to the NRMSEs depicted in (c) and (d). A clear shift of the entire distribution to lower (higher) values is visible.

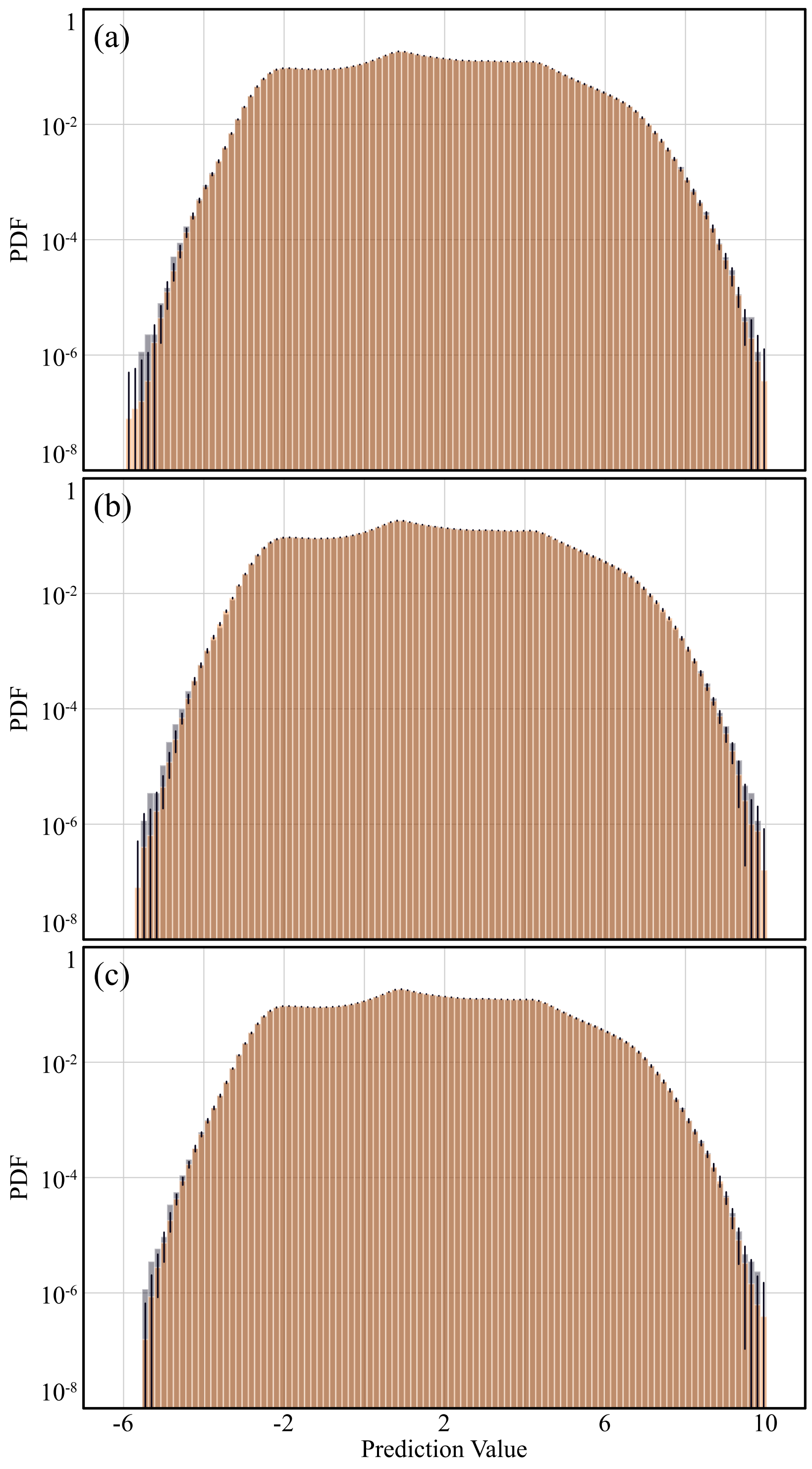

Figure 5.9: PDFs for the longterm predictions of the L96 system. The PDFs are estimated via two superimposed 100 bin histograms of the simulated (gray) and the predicted (orange) data of the **(a)** SN, **(b)** CC and **(c)** MI Neighborhoods. Each entry in the histogram is generated by the prediction/simulation of a single dimension value, with the dataset coming from three 1000 Lyapunov time long sequences. The errorbars represent the 1σ standard deviation of the predicted histogram bins.

types. This is one more small surprise as the comparatively worse L96 CC NRMSE does not seem to effect the long term PDFs at all.

Lyapunov Exponents

Using the OS method introduced in section 4.3 we can calculate the MLEs of the predictions as another characteristic to quantify long term prediction accuracy.

We base the OS calculation on the same three 1000 Lyapunov time long term prediction datasets that generated the PDFs in the previous section. For each of the 30 realizations we choose 100 trajectory positions uniformly distributed in the first of the three long term datasets as starting point for the orbit separation. In the next step, we synchronize the reservoirs that originally created the data for at least 2000 time steps before each starting point.

Once reached, we add normally distributed noise with standard deviation $\sigma_{\text{noise}} = 1 \times 10^{-10}\sigma_{\text{r}}$ to the internal reservoir states. From this perturbed internal state, we let the reservoir predict 1500 time steps and compute the separation magnitude to the unperturbed predicted time series. This is repeated for each of the three long term datasets for each of the 30 realizations for a total of 9000 measured separation magnitudes.

Lastly, we again use a linear least squares fit to calculate the MLEs listing the results in table 5.4. As with the PDFs, the MLEs show excellent agreement with the MLEs calculated directly from the simulations. Notably, this is true even for the SN and MI neighborhoods

System	SIM	SN	CC	MI
KS	0.049	0.046 ± 0.004	0.048 ± 0.002	0.048 ± 0.002
L96	0.45	0.43 ± 0.05	0.47 ± 0.04	0.44 ± 0.04

Table 5.3: MLEs calculated via the OS method for the simulated (SIM) and predicted trajectories using the SN, CC and MI neighborhoods for the KS and L96 data. The errors represent the 1σ standard deviation between network realizations.

in the KS system with the only noticeable difference being a slightly larger variance for the SN neighborhood. Therefore, due to the relatively low impact on the MLE, the previously mentioned behavior of the prediction getting "stuck" in a part of the attractor, might have never been discovered without calculating the PDFs. This highlights the need for multiple analysis methods when quantifying the climate reproduction of a prediction.

5.4 Combined System Prediction

To test the usefulness of GLS for non-locally interacting systems we use the KS and L96 systems to artificially create a non-locally interacting test system. As depicted in Figure 5.10

we do this by concatenating both systems and then randomly shuffling the 80 dimensions of the combined system. As this combined system now is a composite of two systems with

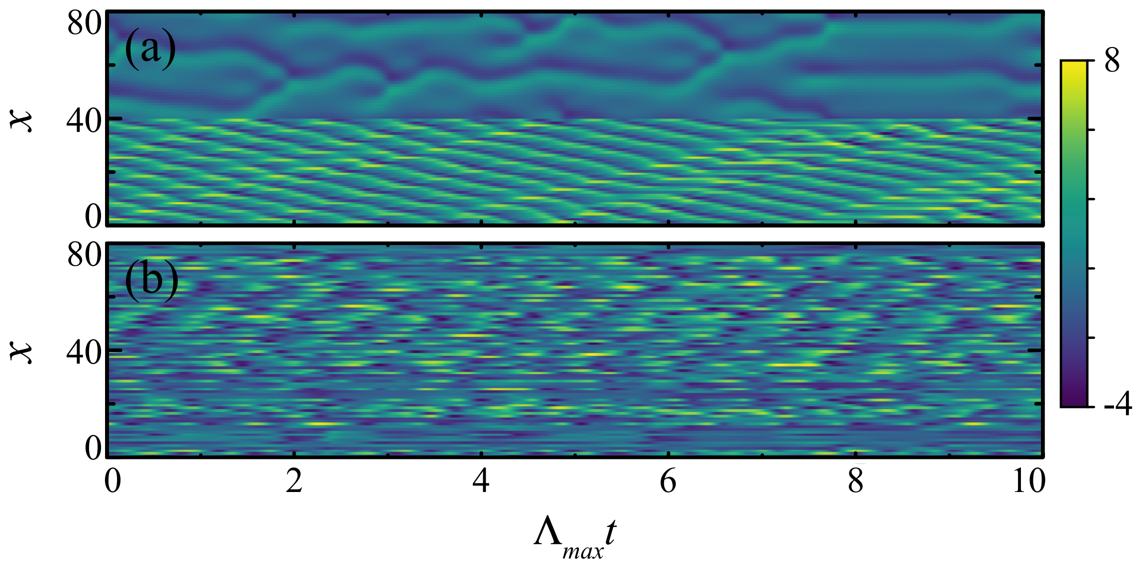

Figure 5.10: Concatenated System **(a)** Simulated data combined from the L96 (dimensions 1-40) and KS (dimensions 41-80) systems. **(b)** Simulated data as in (a) with the dimensions shuffled. The time axis is scaled by the MLE Λ_{max} of the KS model.

different time steps, it does not have a well defined Lyapunov exponent any more. For the sake of consistency we nonetheless continue the time axis rescaling in terms of Lyapunov exponents. For this we now using the larger of the two system's time steps per Lyapunov time as calculated in table 5.1, hence at worst slightly underestimating our short term prediction results in section 5.4.1.

As before, we can also calculate the SN, CC and MI neighborhoods for this new concatenated system. The CC and MI neighborhoods are shown in Figure 5.11. The SN neighborhoods have been omitted from this Figure as they are, by definition, always the same. Fascinatingly, both the CC and MI neighborhoods in the combined but not shuffled systems look like they are composed of the individual system's neighborhoods. In fact, exactly this is the case as both the CC and the MI SMs are able to completely separate the KS and L96 systems.

As will become crucial in the next section, the neighborhoods for the shuffled system have been calculated from shuffled data directly and not, as one might assume, been calculated for the un-shuffled combined system and then shuffled in tandem with the simulated data.

5.4.1 Short Term Predictions

Using the same procedure as in section 5.3.2 we quantify the short term prediction accuracy using the NRMSE. The results are shown in Figure 5.12. Immediately noticeable is the almost instant divergence of all SN predictions. This is of course expected, considering the

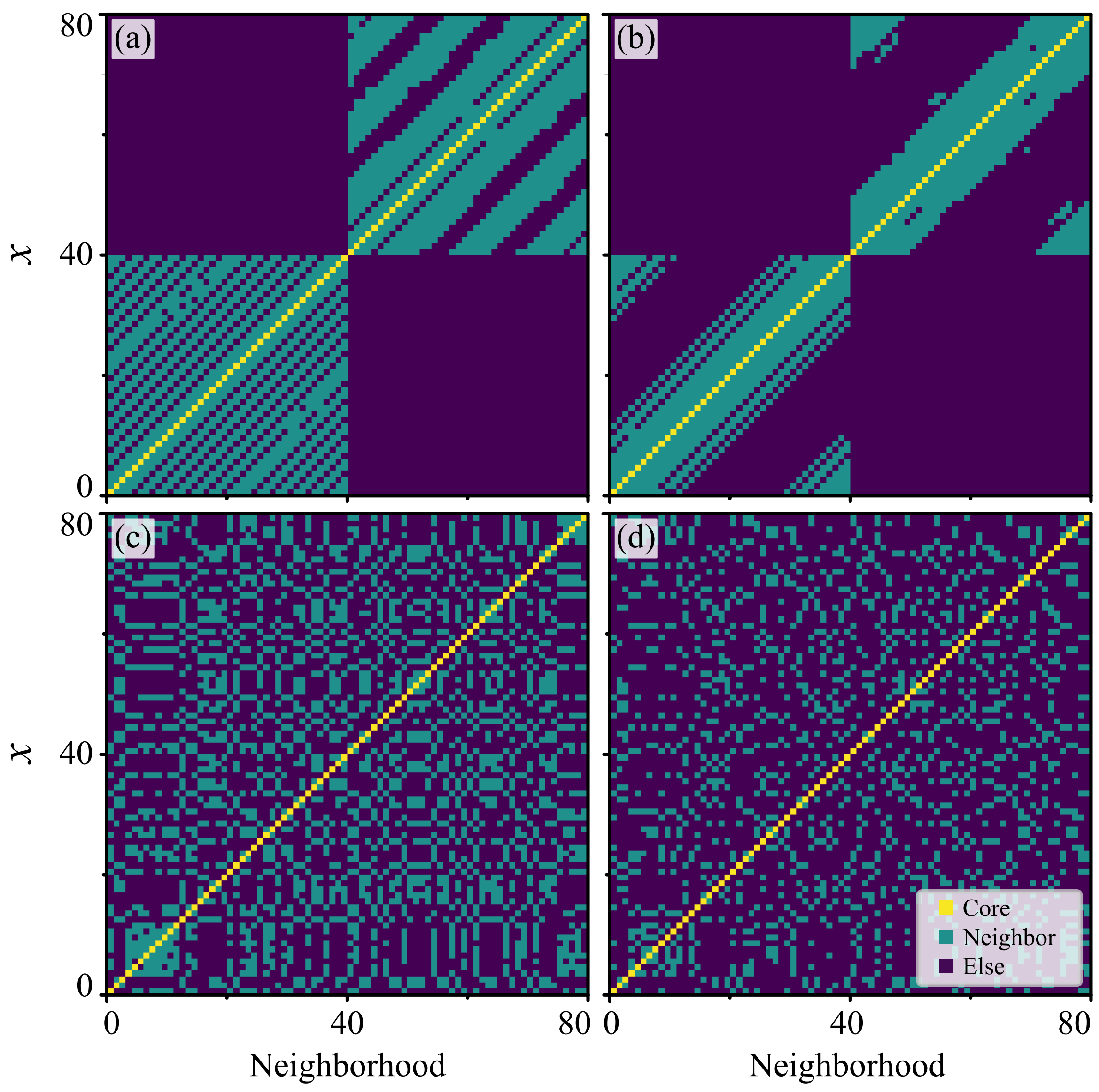

Figure 5.11: Neighborhoods of the concatenated L96 and KS systems. **(a)** is the CC and **(b)** the MI neighborhoods for the concatenated, un-shuffled system. Dimension 1-40 of the data used to compute these neighborhoods come from the L96 system with dimensions 41-80 originating from the KS simulation In **(c)** the CC and **(d)** the MI neighborhoods for the shuffled system are shown.

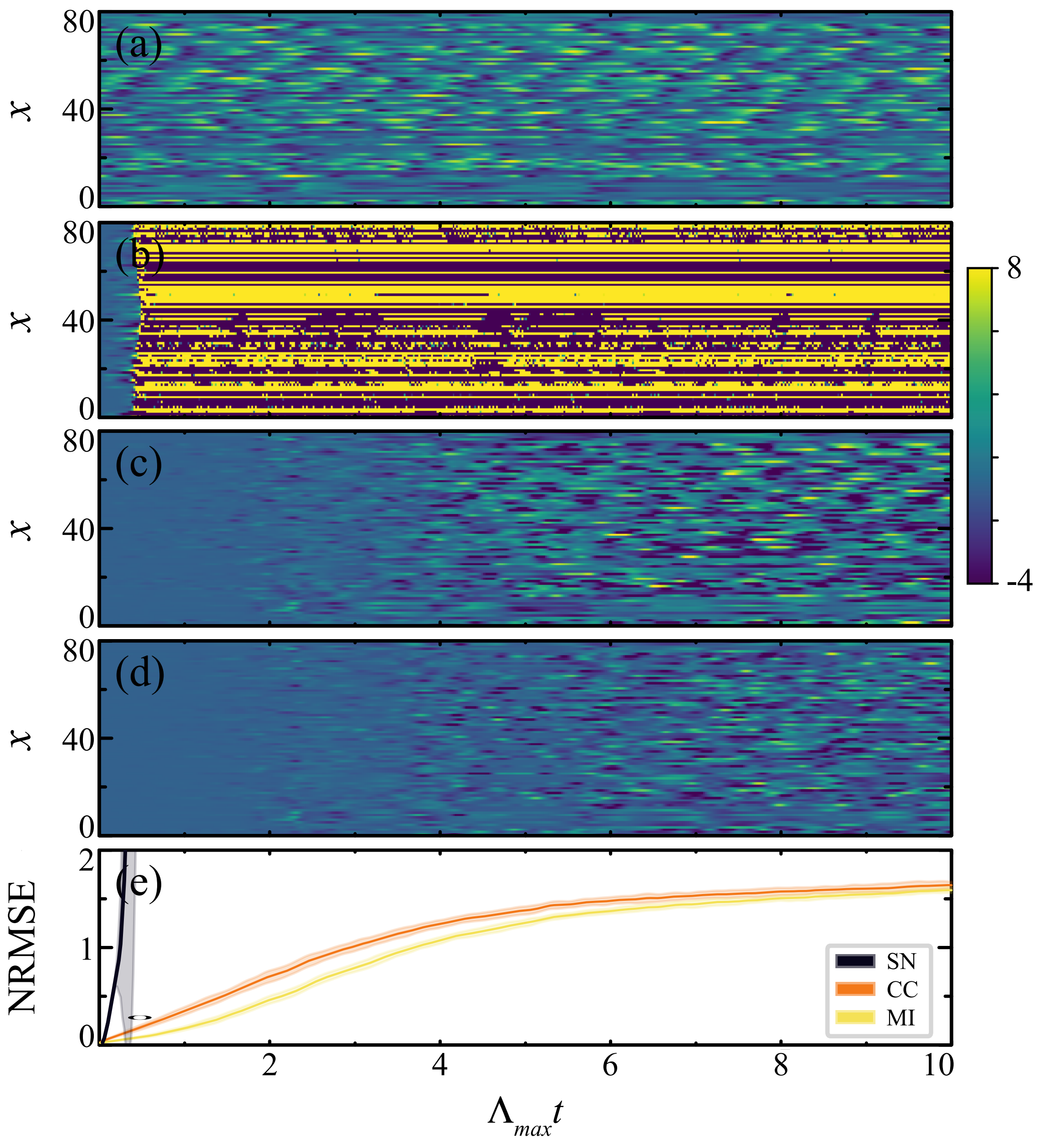

Figure 5.12: Shuffled system short prediction comparison. **(a)** Simulated data of the combined shuffled KS-L96 system. **(b-d)** Exemplary error in the RC prediction when using the **(b)** SN, **(c)** CC and **(d)** MI neighborhoods. The color scale of the diverging prediction was cut to the ensure legibility of the other plots **(e)** NRMSE of SN, CC, and MI prediction data averaged over first the 300 predicted sections and then the 30 network realizations. The error bands correspond to the 3σ standard deviation of the random network realizations.

SN neighborhood's assume a locally interacting system which the shuffled system is not. Furthermore the CC and MI neighborhood's NRMSE is the combination of the individual system's NRMSE which again, makes sense due to the perfect separation between the L96 and the KS system shown in Figure 5.11.

5.4.2 Climate Reproduction

For the long term statistical analysis we also use the same procedure as before with the exception of omitting the MLE calculation which, as mentioned, is not well defined due to the different time step sizes of the sub-systems. The CC and MI PDFs are depicted in Figure 5.13. Again, the PDFs show excellent agreement. From the neighborhoods depicted in Figure 5.11 this could have been predicted as the complete separation of the two subsystems means that the corresponding RCs also do not interact. Hence the computed histograms for the combined system are, in this case literally, the added histograms of the independent sub-system predictions.

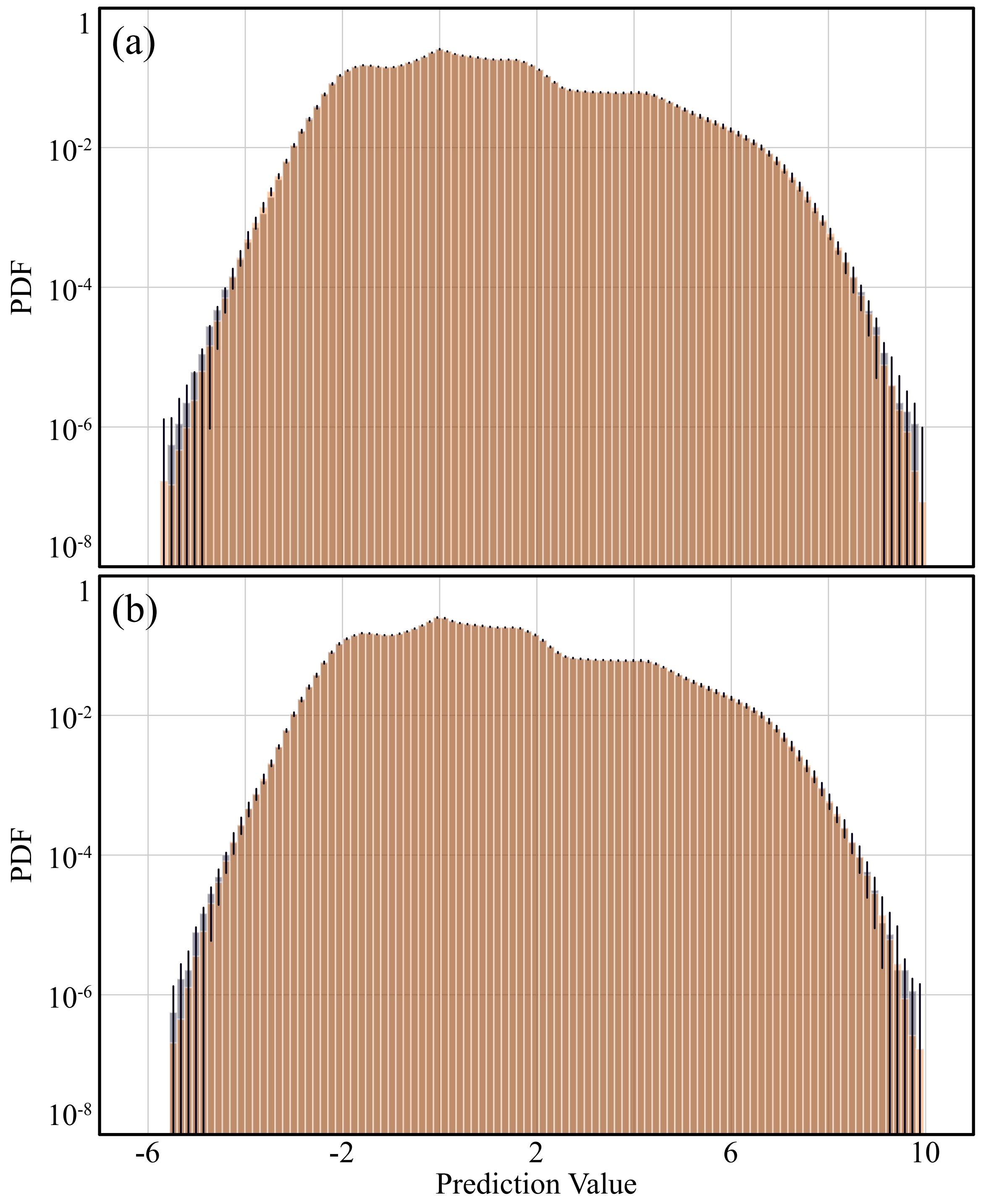

Figure 5.13: PDFs for the longterm predictions of the combined and shuffled KS-L96 system. The PDFs are estimated via two superimposed 100 bin histograms of the simulated (gray) and the predicted (orange) data of the **(a)** CC or **(b)** MI Neighborhood. Each entry in the histogram is generated by the prediction of a single dimension value, with the dataset coming from the prediction of three 1000 Lyapunov time long sequences. The error bars represent the 1σ standard deviation of the predicted histogram bins.

5.5 GLS Neighborhoods in higher Dimensions

As GLS is to a large part a complexity reduction method, its scalability to higher dimensional systems than considered so far is essential for its wide spread usefulness. To demonstrate this scaling behavior one could repeat the preceding analyses for very high dimensional input but, due to the large amount of computation time required, this is outside the scope of this thesis.

Considering that we have shown RC being more than capable of learning and predicting the L96 and KS systems using the GLS neighborhoods, we also do not need to do so, as long as we can demonstrate the higher dimensional neighborhoods to have the same properties allowing the CC and MI neighborhoods to create high quality prediction in the 40 dimensional system. Namely that, for locally interacting systems, the core's nearest spatial neighbors are included in the GLS neighborhoods.

Therefore, we will demonstrate the scalability of GLS to higher dimensional input by simulating the KS and L96 equations using a larger system dimension of $D = 400$. For this, we keep all other system parameters, notably L96's forcing $F = 5$ as well KS's system size $L = 22$, unchanged.

We use OS to calculate the MLEs of the higher dimensional systems. As listed in table 5.4 we find them to again be chaotic, making them suitable example systems.

System	$D = 40$	$D = 400$
KS	0.049	0.047
L96	0.45	0.62

Table 5.4: MLEs calculated via the OS method for the simulated KS and L96 of different dimension D. All other system parameters of the L96 and KS systems remain unchanged.

Generating the neighborhoods as before, we find that the MI neighborhoods do not need any adjustments to accurately reproduce the local neighborhoods around each core. The CC neighborhoods on the other hand need a slightly increased neighborhood size of 33 compared to the 27 of the 40 dimensional L96 to also include the core's nearest neighbors which, as discussed, is paramount for a working prediction. The resulting neighborhoods for the individual, as well as for the combined systems are shown in Figure 5.14

Even though the CC neighborhood elements include dimensions much farther out from the core than in the MI neighborhoods, both are still very much clustered around the cores. Furthermore, we note that neighborhoods for the combined system show that the separation between the two sub systems works even for very large dimensions. Crucially though, the adjustment needed for the CC neighborhood to reconstruct the core's local neighborhoods including its nearest neighbors is minimal. For the MI SMs no adjustment was needed at all.

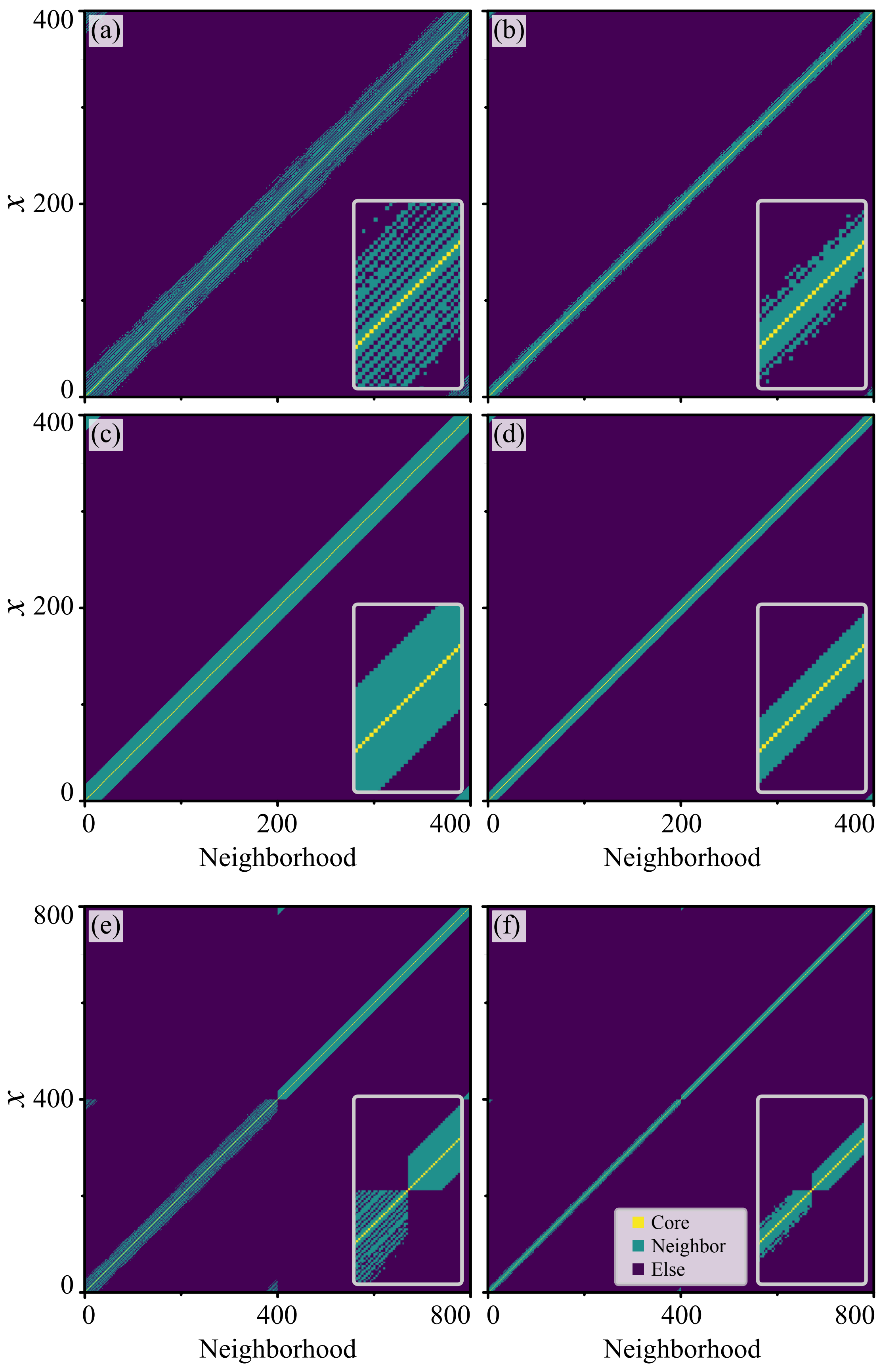

Figure 5.14: CC and MI neighborhoods for **(a)-(b)** the 400 dimensional L96, **(c)-(d)** the 400 dimensional KS and **(e)-(f)** the 800 dimensional combined system. (a), (c) and (e) are CC while (b), (d) and (f) are MI neighborhoods. The insets are magnifications of the center of each plot.

Combining the knowledge that this reproduction of the local neighborhoods is usually the key for predicting locally interacting systems, or at least is the key for the KS and L96 systems studied here, with the results from Pathak et. al. [4], predicting KS systems of this dimension and larger using similarly sized SN neighborhoods, gives a strong indication that at least for the case of the individual and combined L96 and KS systems, scaling GLS to large dimensions works.

5.6 Clustering

To see if clustering of cores can be achieved with GLS we start by again looking at the individual L96 and KS systems, simulated as in section 5.3.

5.6.1 Locally Interacting Systems

The single core neighborhoods are computed as discussed in section 3.3.1. Additionally, we also divide the 40 dimensional system into neighborhoods with an average of two, four and eight core dimensions, resulting in a total of 20, 10 and 5 total neighborhoods.

For the SN neighborhoods this is easily done by taking adjacent sets of input dimensions as cores and the spatially closest remaining dimensions as neighbors. The result of this process is depicted in Figure 5.15.

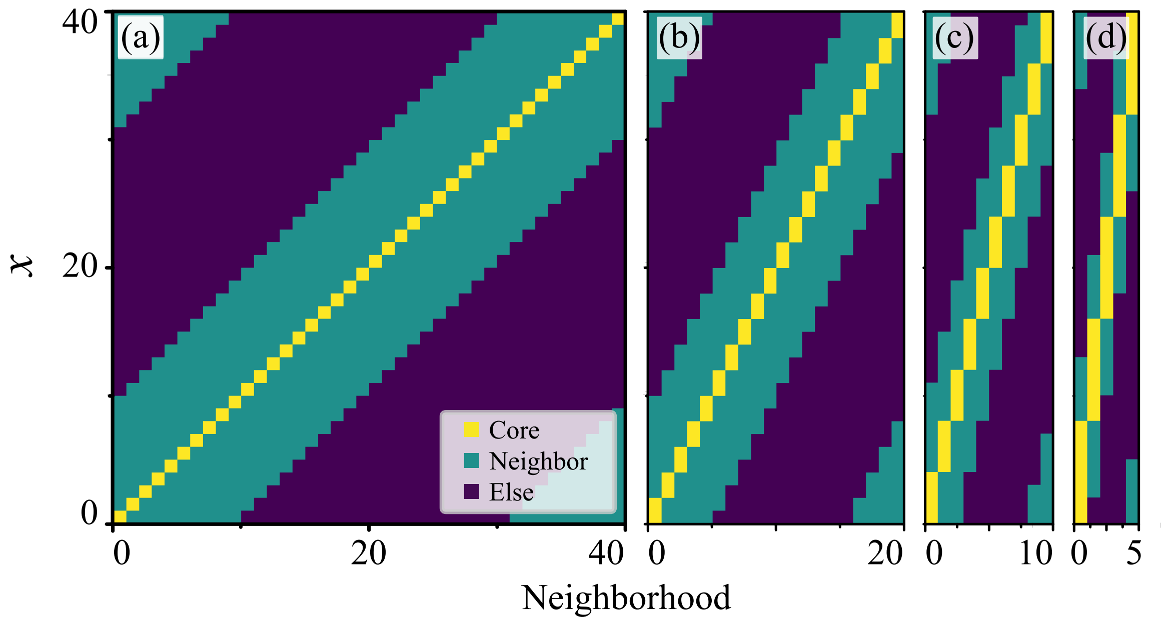

Figure 5.15: SN Neighborhoods for a 40 dimensional system with **(a)** one **(b)** two **(c)** four **(d)** eight cores per neighborhood. The total neighborhood size of 19 is the same for all cases.

For the CC and MI neighborhoods we use the agglomerative clustering algorithm described in section 3.3.2 resulting in the neighborhoods depicted in Figures 5.16 and 5.17. All of the

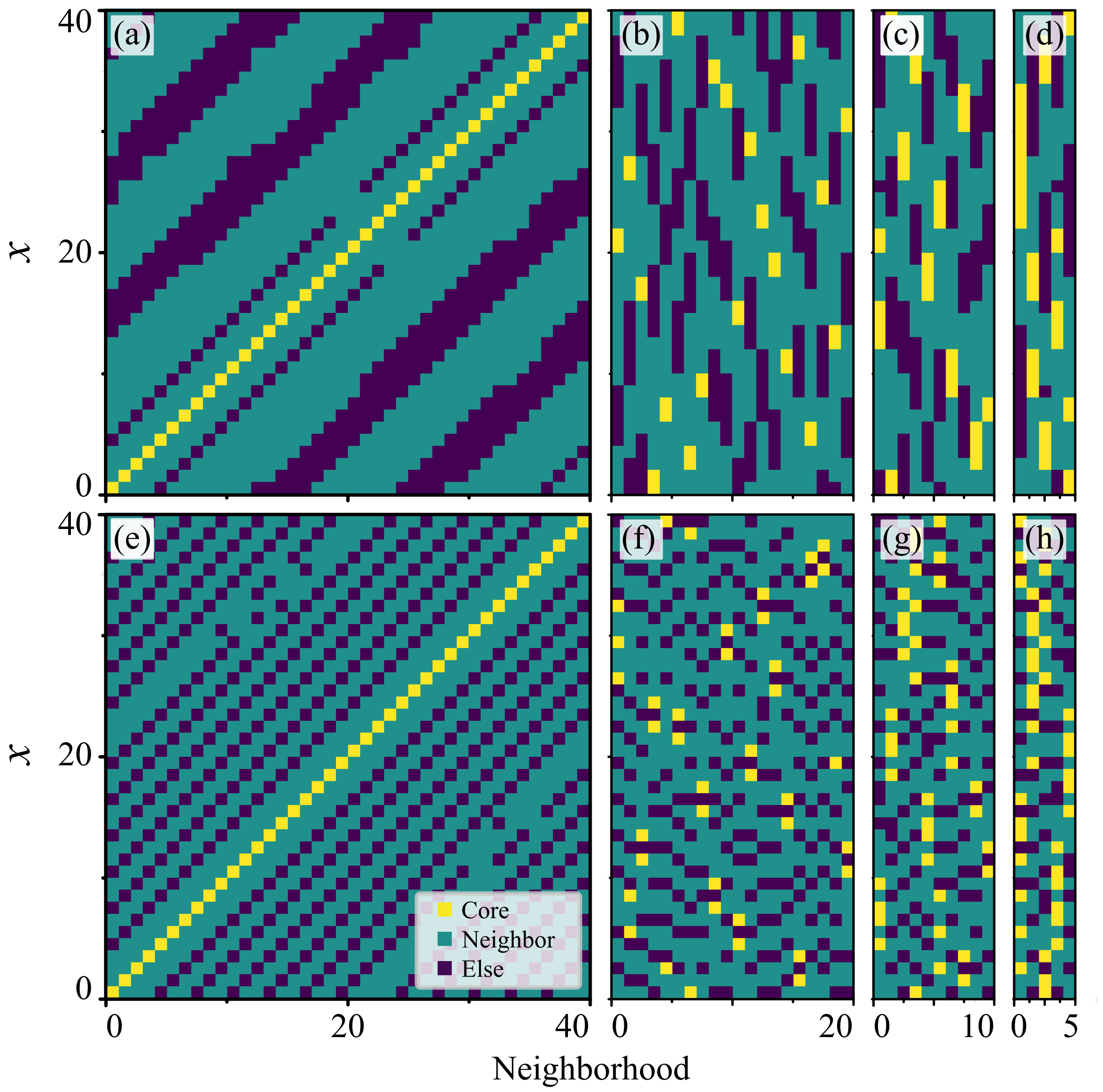

Figure 5.16: Clustered CC neighborhoods for the **(a)-(d)** KS and **(e)-(h)** L96 systems. The neighborhoods have an average of one (a, e), two (b, f) four (c,g) or eight (d, h) cores.

computed neighborhoods include the dimensions around each core as neighbor, with the only exception being the clustered CC neighborhoods for the L96 system. (Figure 5.16f-h). As the systems simulated here are all locally interacting systems, this failure to reproduce the local neighborhood around the cores inevitably leads to a failed prediction as we discussed during section 5.3.1 .

Nonetheless, we can again use these neighborhoods to compute the short and long term predictions for the L96 and KS systems using the same procedure and hyperparameters as in section 5.3. The short term prediction results are given in Figures 5.18, 5.19 and 5.20.

Starting with the obvious outlier of the L96 CC predictions of Figure 5.19b we see that the predictions diverge quickly for all cases with more than one core. As mentioned, this is a

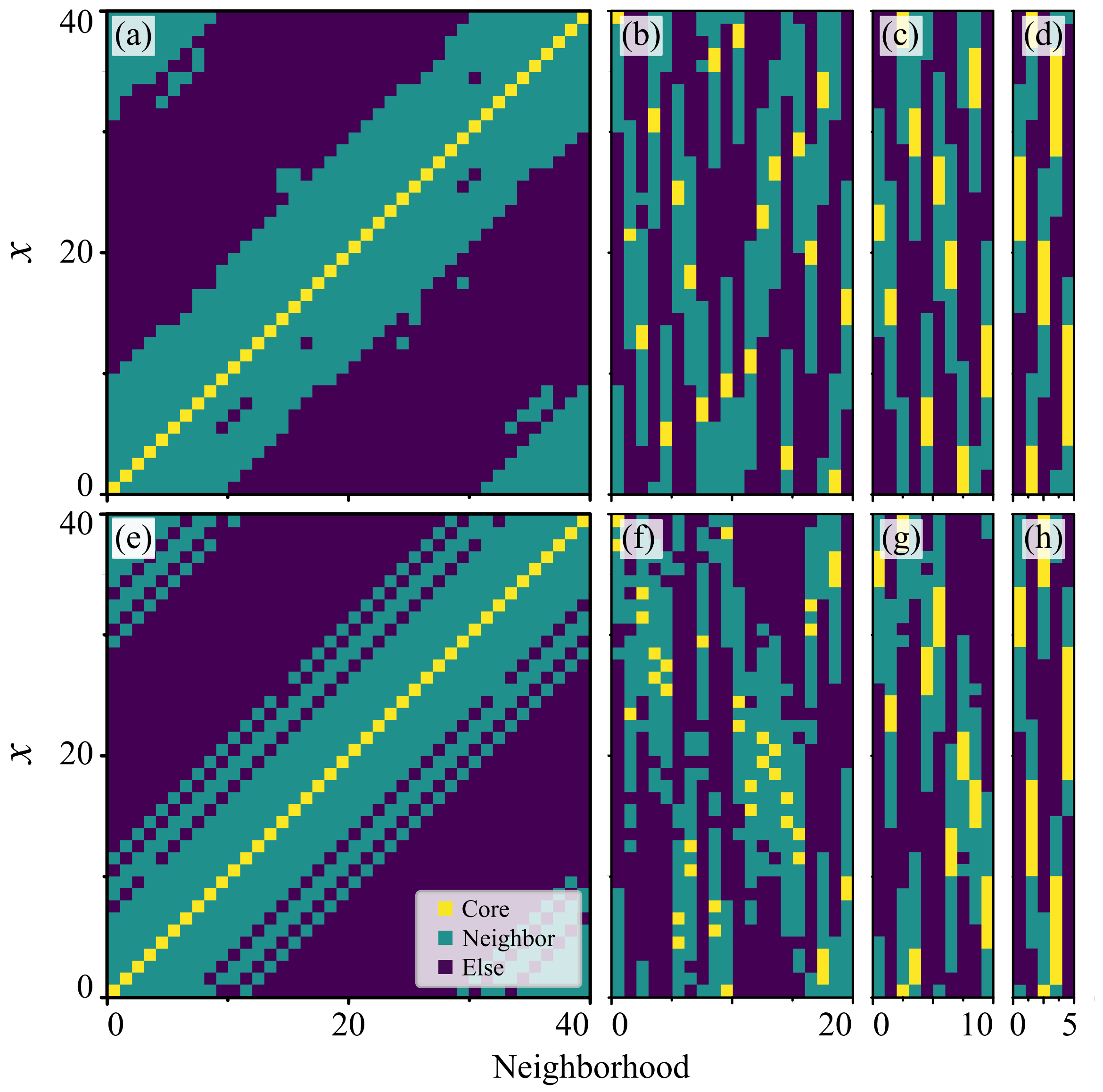

Figure 5.17: Clustered MI neighborhoods for the **(a)**-**(d)** KS and **(e)**-**(h)** L96 systems. The neighborhoods have an average of one (a, e), two (b, f) four (c,g) or eight (d, h) cores

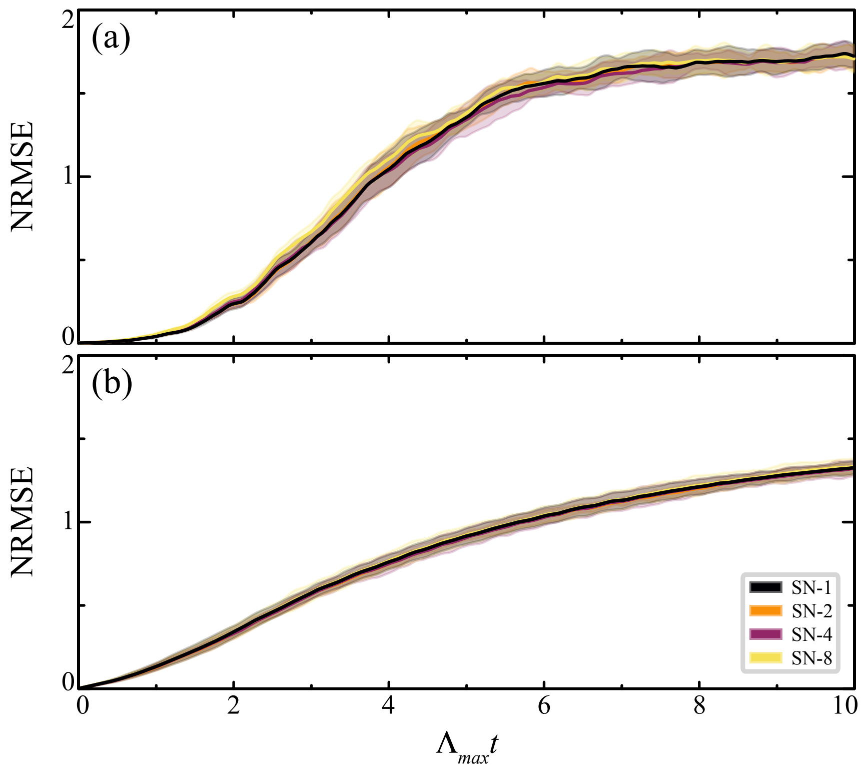

Figure 5.18: SN short term predictions for neighborhoods with an average of one (SN-1), two (SN-2), four (SN-4), and eight (SN-8) cores for the **(a)** KS and **(b)** L96 systems. The error bands correspond to the 3σ standard deviation of the random network realizations.

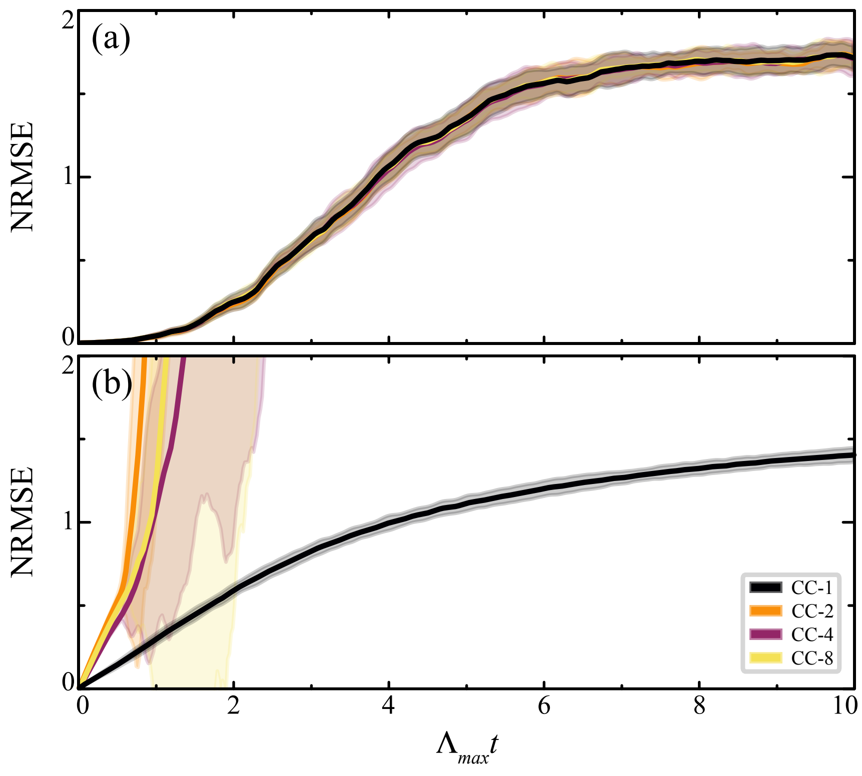

Figure 5.19: CC short term predictions for neighborhoods with an average of one (CC-1), two (CC-2), four (CC-4), and eight (CC-8) cores for the **(a)** KS and **(b)** L96 systems. The error bands correspond to the 3σ standard deviation of the random network realizations.

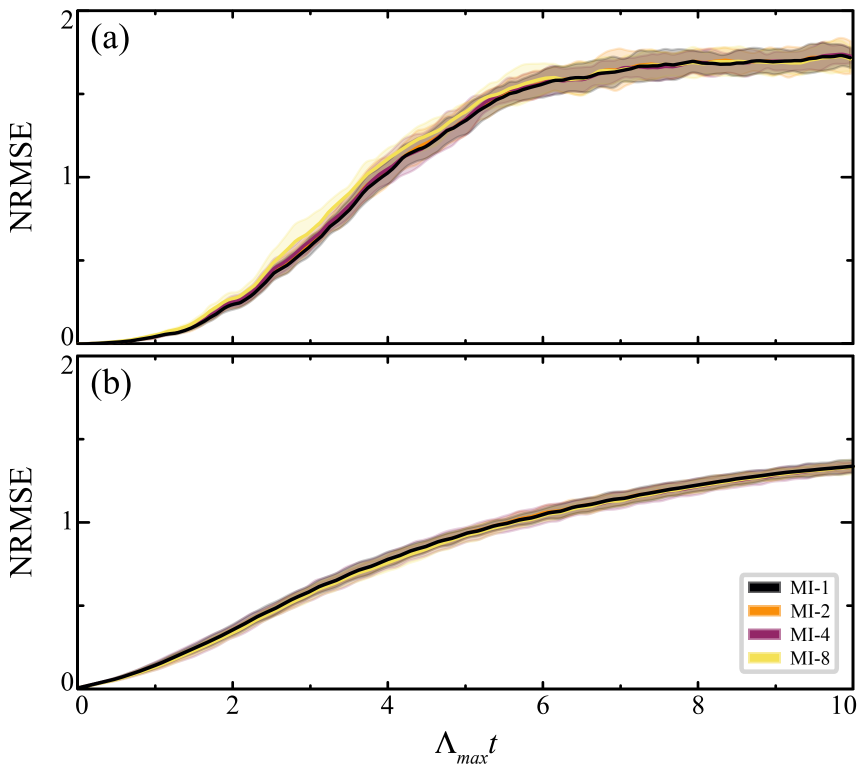

Figure 5.20: MI short term predictions for neighborhoods with an average of one (MI-1), two (MI-2), four (MI-4), and eight (MI-8) cores for the **(a)** KS and **(b)** L96 systems. The error bands correspond to the 3σ standard deviation of the random network realizations.

case of the core's local nearest neighbors not being included in the neighborhood. All other clustered neighborhoods achieve the same performance as their non-clustered counterparts.

Calculating the MLEs in the same manner as previously we find that, with exception of the diverging L96 CC neighborhoods, all calculated MLEs agree very well with the simulated values. The MLEs are listed in table 5.5.

Additionally, the PDFs of the clustered SN and MI do not exhibit the same shift their single core counterparts do (see section 5.3.3) which is further reflected in the decreased variance of SN-2, SN-4 and SN-8 compared to SN-1.

The results are notable, not only because it is evident that the agglomerative clustering algorithm works as intended, but also because the neighborhoods with more cores are much faster to train and predict with. This speed-up is a direct result of the parallelizability as mentioned in section 3.1. Theoretically, neighborhoods with an average of eight cores could be up to eight times faster than their single core counterparts while achieving at least equivalent performance.

Finally, while neighborhood sizes exist for which the CC neighborhoods deliver accurate predictions, in practice they turn out to be too large to be useful. For e.g. CC-8 and the 40 dimensional L96 predicted here, this neighborhood size turns out to be 39. Therefore, we

System	SIM	SN-1	SN-2	SN-4	SN-8
KS	0.049	0.046 ± 0.004	0.047 ± 0.002	0.049 ± 0.002	0.048 ± 0.002
L96	0.45	0.43 ± 0.05	0.42 ± 0.04	0.43 ± 0.04	0.44 ± 0.04

System	SIM	CC-1	CC-2	CC-4	CC-8
KS	0.049	0.048 ± 0.002	0.048 ± 0.002	0.050 ± 0.002	0.049 ± 0.003
L96	0.45	0.47 ± 0.04	–	–	–

System	SIM	MI-1	MI-2	MI-4	MI-8
KS	0.049	0.048 ± 0.002	0.048 ± 0.001	0.049 ± 0.002	0.050 ± 0.003
L96	0.45	0.44 ± 0.04	0.43 ± 0.02	0.43 ± 0.02	0.44 ± 0.02

Table 5.5: MLEs calculated via the OS method for the simulated (SIM) and predicted trajectories of the KS and L96. The predictions are grouped by the similarity measure (SN, CC or MI) and the average number of core dimensions present in each neighborhood (1, 2, 4 or 8). The errors represent the 1σ standard deviation between network realizations.

kept the neighborhood size of 29 here and in the following, as it more accurately showcases the problems of using CC as a SM.

5.6.2 Combined Systems Clustering

Considering the promising results achieved by agglomerative clustering for the individual systems, expecting similarly good predictions when applying clustering to the combined system from section 5.4 is very reasonable. Somewhat counterintuitively, agglomerative clustering for the combined system does not work. As can be seen in Figure 5.21 the clustering of neighborhoods for both MI and CC results in a very uneven distribution of cores between the neighborhoods.

While in principle still better than randomly distributing dimensions between neighborhoods, in practice it does not make a difference as, with the exception of some 40 dimensional MI neighborhoods, most predictions using these neighborhoods diverge. This result is fundamentally tied to how agglomerative clustering works, namely that there is no limit to the cluster size it can create. While uneven cluster sizes can be an advantage, e.g. when the data is composed of smaller, internally interacting subgroups of different sizes, it can also lead to results as shown here.

5.6.3 Higher Dimensional clustered Neighborhoods

Using the same 400 dimensional systems and neighborhood sizes as in section 5.5 we compute the two, four and eight core neighborhoods resulting in Figure 5.22.

For the 400 dimensional individual systems the clustered neighborhoods exhibit the same behavior as for the 40 dimensional clustering, namely that, besides the L96 CC clustering, all

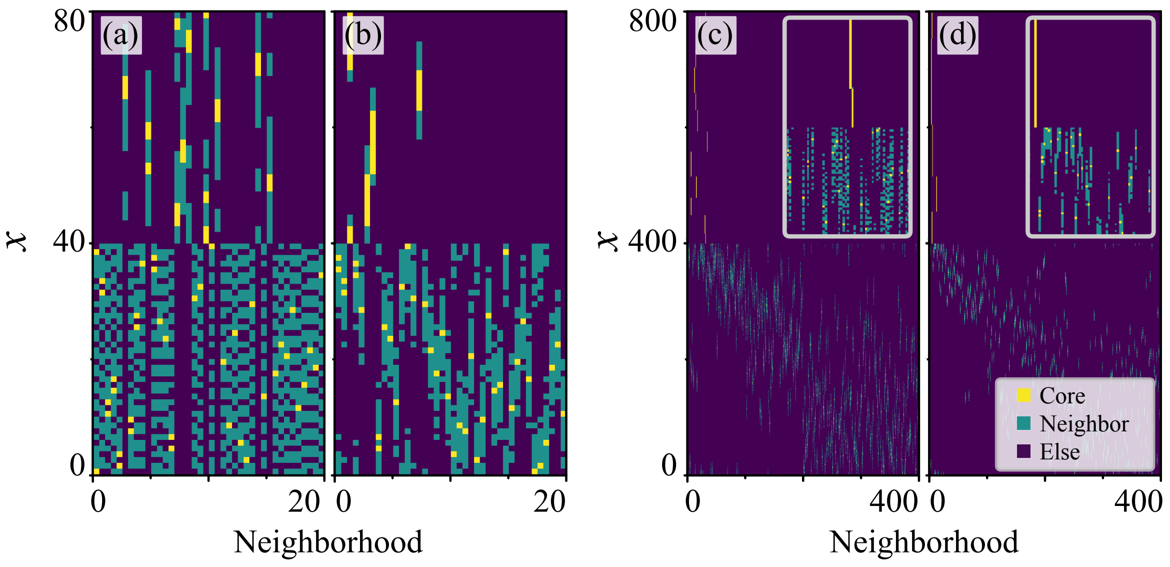

Figure 5.21: CC-2 and MI-2 neighborhoods for **(a)-(b)** the 40 dimensional and **(c)-(d)** the 400 dimensional combined L96-KS. (a) and (c) are CC, (b) and (d) are MI neighborhoods. The insets in (c) and (d) are magnifications of the left center area of each plot.

neighborhoods consist of a mostly cohesive group of cores surrounded by neighbors, which as discussed is the main condition for a good prediction of the locally interacting systems studied here. This gives a strong indication that the clustering of L96 and KS is generalizable to arbitrary dimensions.

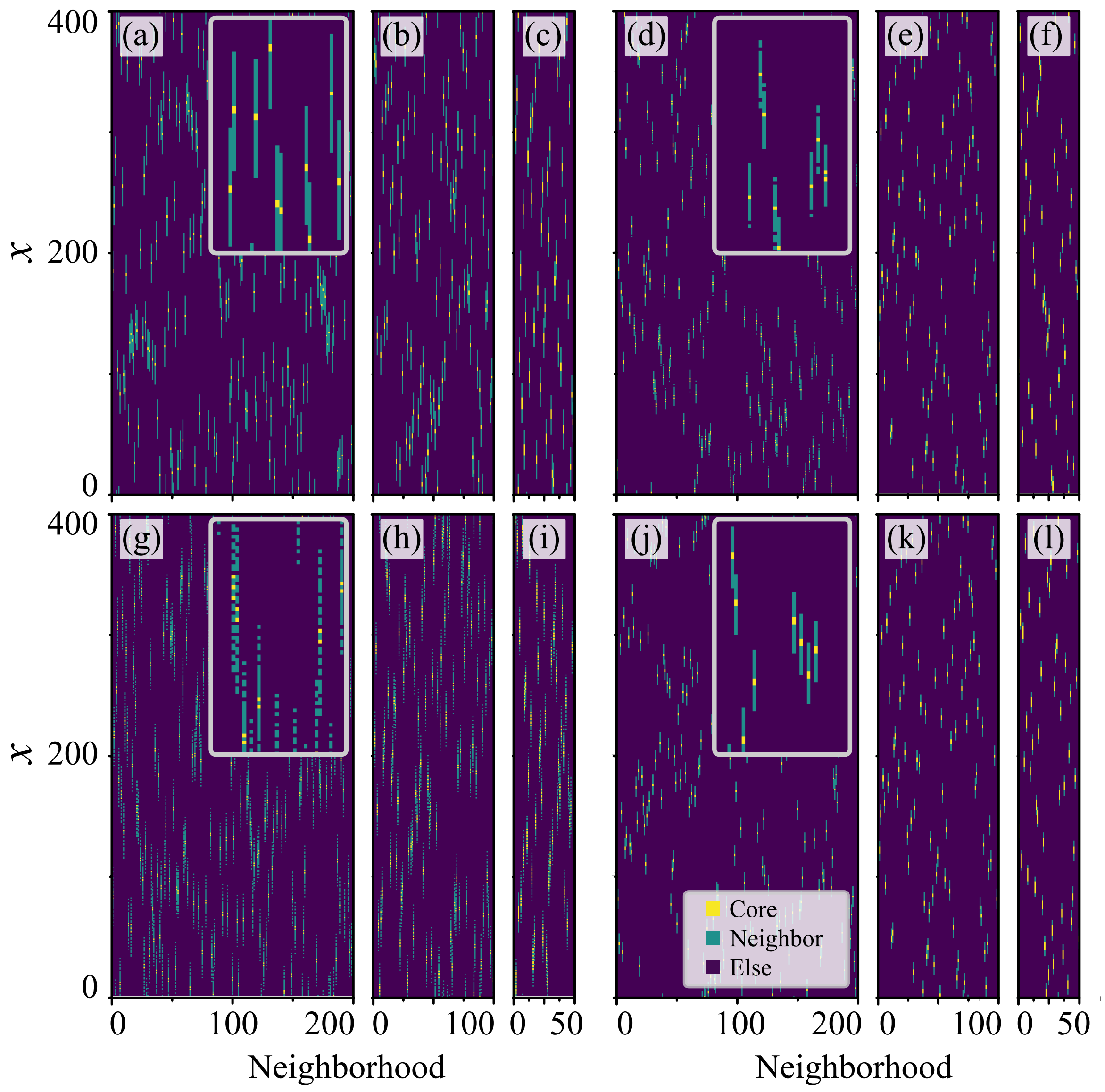

Figure 5.22: CC and MI neighborhoods for the 400 dimensional **(a)-(f)** KS and **(g)-(l)** L96 systems. (a)-(c) and (g)-(i) are CC, while (d)-(f) and (j)-(l) are MI neighborhoods. The insets in (a), (d), (g) and (j) are magnifications of representative parts of the neighborhoods.

6. Conclusion and Outlook

This thesis introduced the method of GLS, an extension of Pathak et al.'s LS [4] approach by generalizing spatial locality to the concept of similarity using the information measures CC and MI.

We demonstrated this approach by using RC to reproduce the short and the long term statistics achieved by LS in the locally interacting L96 and KS systems without explicitly using this local interaction.

Furthermore, we created a non-locally interacting system by first concatenating and then shuffling the dimensions of L96 and KS. We then accurately predicted this system using GLS while the conventional LS approach failed completely.

In the next step we further improved the scalability of GLS by using agglomerative clustering to group the most similar input dimensions as cores of the same neighborhoods. While agglomerative clustering was not able to generate useful neighborhoods for the mixed L96-KS system, it accurately grouped the dimensions in three out of four cases, only failing when applying the CC SM to the L96 system.

Additionally, we demonstrated the successful scaling of single core and clustered GLS neighborhoods to high dimensions in all systems where they were able to create accurate predictions in the lower dimensional case.

We want to again emphasize that the working principle of the GLS approach does not depend on our choice of ML technique, illustrative systems, or SMs. Rather, GLS is a very broad concept, allowing the grouping of time series by their similarity and using these groupings to infer the future of said time series.

Nonetheless, while substantial results have been achieved, multiple new opportunities and potential improvements were uncovered during this thesis. First and foremost, GLS has yet to be applied on real experimental data with a non-local interaction structure. Such data with purely abstract connections can for example be found in financial markets.

The choice of SM is another aspect that has yet to be investigated in more detail. While this thesis has shown that the MI delivers consistently better results than the CC, many more choices as SM are possible. Causal measures, like the transfer entropy, or one of the many extensions of MI could allow a significant increase in the SM's efficacy when calculating the most relevant neighbors for any given core. In the analysis of more complicated or noisy data, this could prove crucial.

Additionally, while we layed the foundation for the clustering of core dimensions, further work in this area seems worthwhile. Not only does clustering come with a significant decrease in computation time, it could also enable a further increase in neighborhood quality by grouping inherently tied dimensions, such as position and velocity of a particle, to be predicted together.

Lastly, expanding the measure of similarity to take changes over time into account, for example in the relationships between humans, could open up yet another new field of problems GLS could be used for.

Bibliography

[1] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.

[2] Harri Valpola. From neural PCA to deep unsupervised learning. In *Adv. Indep. Compon. Anal. Learn. Mach.*, pages 143–171. Elsevier, jan 2015.

[3] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*, volume 19, 2014.

[4] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach. *Phys. Rev. Lett.*, 120(2):24102, 2018.

[5] Herbert Jaeger. The " echo state " approach to analysing and training recurrent neural networks. *GMD Rep.*, (148):1–47, 2001.

[6] Pantelis R. Vlachas, Jaideep Pathak, Brian R. Hunt, Themistoklis P. Sapsis, Michelle Girvan, Edward Ott, and Petros Koumoutsakos. Forecasting of Spatio-temporal Chaotic Dynamics with Recurrent Neural Networks: a comparative study of Reservoir Computing and Backpropagation Algorithms. 2100:0–2, 2019.

[7] Ashesh Chattopadhyay, Pedram Hassanzadeh, and Devika Subramanian. Data-driven prediction of a multi-scale Lorenz 96 chaotic system using deep learning methods: Reservoir computing, ANN, and RNN-LSTM. 2019.

[8] Edward N. Lorenz. Predictablilty: A problem partly solved, 1996.

[9] Y. Kuramoto and T. Tsuzuki. Persistent Propagation of Concentration Waves in Dissipative Media Far from Thermal Equilibrium. *Prog. Theor. Phys.*, 55(2):356–369, feb 1976.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q.

Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[12] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.*, 3(3):127–149, aug 2009.

[13] Ray J Frank, Neil Davey, and Stephen P Hunt. Time series prediction and neural networks. *Journal of intelligent and robotic systems*, 31(1-3):91–103, 2001.

[14] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[16] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–2560, nov 2002.

[17] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent Advances in Physical Reservoir Computing: A Review. 2019.

[18] T. L. Carroll and L. M. Pecora. Network structure effects in reservoir computers. *Chaos*, 29(8), 2019.

[19] Alexander Haluszczynski and Christoph Räth. Good and bad predictions: Assessing and improving the replication of chaotic attractors by means of reservoir computing. *Chaos*, 29(10), 2019.

[20] Zhixin Lu, Brian R. Hunt, and Edward Ott. Attractor reconstruction by machine learning. *Chaos*, 28(6):061104, jun 2018.

[21] C. E. Shannon. A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, 27(4):623–656, 1948.

[22] C. J. Cellucci, A. M. Albano, and P. E. Rapp. Statistical validation of mutual information calculations: Comparison of alternative numerical algorithms. *Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys.*, 71(6):066208, jun 2005.

[23] Young Il Moon, Balaji Rajagopalan, and Upmanu Lall. Estimation of mutual information using kernel density estimators. *Phys. Rev. E*, 52(3):2318–2321, sep 1995.

[24] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Phys. Rev. E - Stat. Physics, Plasmas, Fluids, Relat. Interdiscip. Top.*, 69(6):16, 2004.

[25] Alexander Haluszczynski, Ingo Laut, Heike Modest, and Christoph Räth. Linear and nonlinear market correlations: Characterizing financial crises and portfolio optimization. *Phys. Rev. E*, 96(6):1–11, 2017.

[26] Alexander Strehl and Joydeep Ghosh. Cluster ensembles - A knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3(3):583–617, 2003.

[27] Xin Jin and Jiawei Han. *K-Means Clustering*, pages 563–564. Springer US, Boston, MA, 2010.

[28] Pankaj Mehta, Marin Bukov, Ching Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. *Phys. Rep.*, 810:1–124, 2019.

[29] Zhixin Lu, Jaideep Pathak, Brian Hunt, Michelle Girvan, Roger Brockett, and Edward Ott. Reservoir observers: Model-free inference of unmeasured variables in chaotic systems. *Chaos*, 27(4), 2017.

[30] Yu Huang, Zuntao Fu, and Christian L. E. Franzke. Detecting causality from time series in a machine learning framework. *Chaos An Interdiscip. J. Nonlinear Sci.*, 30(6):063116, 2020.

[31] Daniel S. Wilks. Effects of stochastic parametrizations in the Lorenz '96 system. *Q. J. R. Meteorol. Soc.*, 131(606):389–407, jan 2005.

[32] Norbert Marwan, Jürgen Kurths, and Saskia Foerster. Analysing spatially extended high-dimensional dynamics by recurrence plots. *Phys. Lett. Sect. A Gen. At. Solid State Phys.*, 379(10-11):894–900, 2015.

[33] William H Press, Saul A Teukolsky, Brian P Flannery, and William T Vetterling. *Numerical recipes in Fortran 77: volume 1, volume 1 of Fortran numerical recipes: the art of scientific computing*. Cambridge university press, 1992.

[34] Yoshiki Kuramoto and Toshio Tsuzuki. Persistent propagation of concentration waves in dissipative media far from thermal equilibrium. *Progress of theoretical physics*, 55(2):356–369, 1976.

[35] Y Pomeau and S Zaleski. The kuramoto-sivashinsky equation: A caricature of hydrodynamic turbulence? In *Macroscopic Modelling of Turbulent Flows*, pages 296–303. Springer, 1985.

[36] S. M. Cox and P. C. Matthews. Exponential time differencing for stiff systems. *J. Comput. Phys.*, 176(2):430–455, 2002.

[37] Aly Khan Kassam and Lloyd N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM J. Sci. Comput.*, 26(4):1214–1233, 2005.

[38] Jaideep Pathak. Model-Free Prediction of Large Spatiotemporally chaotic systems from data - supp material. *Phys. Rev. Lett.*, 3(4):381–420, 2018.

[39] M Jardak, IM Navon, and M Zupanski. Comparison of sequential data assimilation methods for the kuramoto–sivashinsky equation. *International journal for numerical methods in fluids*, 62(4):374–402, 2010.

[40] Fei Lu, Kevin K Lin, and Alexandre J Chorin. Data-based stochastic model reduction for the kuramoto–sivashinsky equation. *Physica D: Nonlinear Phenomena*, 340:46–57, 2017.

[41] Alan Wolf, Jack B Swift, Harry L Swinney, and John A Vastano. Determining Lyapunov exponents from a time series. *Phys. D Nonlinear Phenom.*, 16(3):285–317, 1985.

[42] Michael T. Rosenstein, James J Collins, and Carlo J De Luca. A practical method for calculating largest Lyapunov exponents from small data sets. *Phys. D Nonlinear Phenom.*, 65(1-2):117–134, 1993.

[43] Henry Abarbanel. *Analysis of observed chaotic data*. Springer Science & Business Media, 2012.

[44] Julien Clinton Sprott and Julien C Sprott. *Chaos and time-series analysis*, volume 69. Citeseer, 2003.

[45] Pantelis R. Vlachas, Wonmin Byeon, Zhong Y. Wan, Themistoklis P. Sapsis, and Petros Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-Term memory networks. *Proc. R. Soc. A Math. Phys. Eng. Sci.*, 474(2213):20170844, may 2018.

[46] `https://anaconda.org/`.
Called on: 2020-09-11.

[47] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

Glossary

ANN artificial neural network. 3, 7, 11

CC cross-correlation. 1, 2, 9–11, 22–28, 32–36, 38–41, 43–47, 49

ETDRK4 exponential time-differencing fourth-order Runge-Kutta. 14

FFN feed forward network. 3, 4

GLS generalized local states. 1, 2, 7, 13, 16, 19, 21, 22, 32, 38, 40, 49, 50

KS Kuramoto-Sivashinsky. 1, 2, 9, 14, 15, 19–26, 28–30, 32–34, 36, 38–47, 49, 55, 58

L96 Lorenz-96. 1, 2, 5, 13, 14, 19–25, 27, 28, 31–34, 36, 38–47, 49, 59

LS local states. 1, 2, 13, 19, 21, 22, 49

MI mutual information. 1, 2, 9–11, 22, 24–28, 32–36, 38–40, 42, 44–47, 49, 50

ML machine learning. 1, 3–5, 49

MLE maximal Lyapunov exponent. 2, 16, 17, 19, 26, 32, 33, 36, 38, 44, 45

NRMSE normalized root mean square error. 24–28, 30, 32, 33, 35, 36, 58, 59

OS orbit separation. 16, 19, 20, 32, 38, 45

PDF probability distribution function. 28–32, 36, 37, 44

RC reservoir computing. 1, 3–5, 7, 13, 16, 19, 21, 25–28, 35, 36, 38, 49

RNN recurrent neural network. 3

SM similarity measure. 1, 2, 9–12, 22, 24, 33, 38, 45, 49, 50

SN spatial neighborhood. 11, 21–28, 30, 32, 33, 35, 36, 40, 44, 45

A. Appendix

A.1 Kuramoto-Sivashinsky Simulation Details

Shown here are a comparison of our KS simulation with the one published by Kassam et.al. [37] in Figure A.1 as well as the python code used to produce it and all other KS simulations of this thesis.

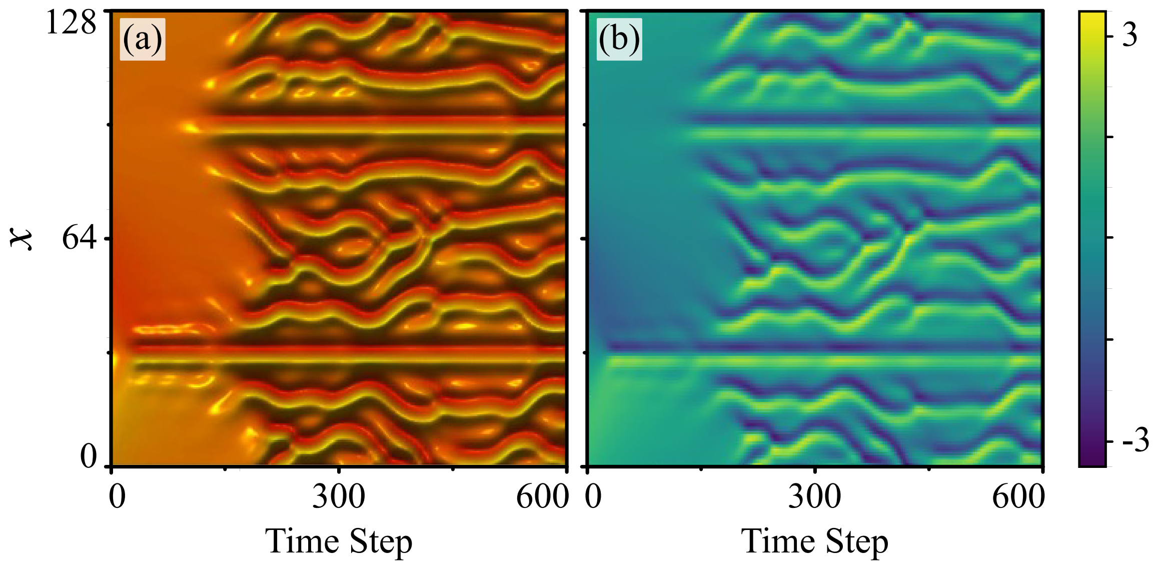

Figure A.1: Comparison of KS simulations using the same initial condition and parameters $N = 128, L = 100, dt = 0.25$. **(a)** Simulation published by Kassam et.al. [37]. The color bar is omitted, as was done in the original paper. Modified from [37] **(b)** Our simulation using the code given in A.1

```python
import numpy as np
def _kuramoto_sivashinsky(dimensions, system_size, dt, time_steps, starting_point):
    """ This function simulates the Kuramoto-Sivashinsky PDE

    The underlying algorithm is called ETDRK4, see Kassam et al. 2005 for details

    Args:
        dimensions (int): nr. of dimensions of the system grid
        system_size (int): physical size of the system
        dt (float): time step size
        time_steps (int): nr. of time steps to simulate
        starting_point (np.ndarray): starting point for the simulation of shape
            (dimensions, )

    Returns:
        (np.ndarray): simulated trajectory of shape (time_steps, dimensions)

    """

    # No. of grid points in real space (and hence dimensionality of the output)
    n = dimensions

    # system size
    size = system_size

    # Define initial conditions and Fourier Transform them
    if starting_point is None:
        # Use the starting point from the Kassam_2005 paper
        x = size * np.transpose(np.conj(np.arange(1, n + 1))) / n
        u = np.cos(2 * np.pi * x / size) * (1 + np.sin(2 * np.pi * x / size))
    else:
        # x = starting_point
        u = starting_point

    v = np.fft.fft(u)

    h = dt  # time step
    nmax = time_steps  # No. of time steps to simulate

    # Wave numbers
    k = np.transpose(
        np.conj(np.concatenate((np.arange(0, n / 2), np.array([0]),
                                np.arange(-n / 2 + 1, 0))))) * 2 * np.pi / size

    L = k ** 2 - k ** 4
    E = np.exp(h * L)
    E_2 = np.exp(h * L / 2)
    M = int(np.ceil(size/(2 * np.pi)))
    r = np.exp(1j * np.pi * (np.arange(1, M + 1) - 0.5) / M)
    LR = h * np.transpose(np.repeat([L], M, axis=0)) + np.repeat([r], n, axis=0)
    Q = h * np.real(np.mean((np.exp(LR / 2) - 1) / LR, axis=1))
    f1 = h * np.real(
        np.mean((-4 - LR + np.exp(LR) * (4 - 3 * LR + LR ** 2)) / LR ** 3,
                axis=1))
    f2 = h * np.real(
        np.mean((2 + LR + np.exp(LR) * (-2 + LR)) / LR ** 3, axis=1))
```

```
f3 = h * np.real(
    np.mean((-4 - 3 * LR - LR ** 2 + np.exp(LR) * (4 - LR)) / LR ** 3,
            axis=1))

# List of Real space solutions, later converted to a np.array
uu = [np.array(u)]

g = -0.5j * k

# ETDRK4 steps in FT space
for n in range(1, nmax + 1):
    Nv = g * np.fft.fft(np.real(np.fft.ifft(v)) ** 2)
    a = E_2 * v + Q * Nv
    Na = g * np.fft.fft(np.real(np.fft.ifft(a)) ** 2)
    b = E_2 * v + Q * Na
    Nb = g * np.fft.fft(np.real(np.fft.ifft(b)) ** 2)
    c = E_2 * a + Q * (2 * Nb - Nv)
    Nc = g * np.fft.fft(np.real(np.fft.ifft(c)) ** 2)

    v = E * v + Nv * f1 + 2 * (Na + Nb) * f2 + Nc * f3
    u = np.real(np.fft.ifft(v))
    uu.append(np.array(u))

uu = np.array(uu)
return uu
```

A.2 Realization vs Starting Position Variance

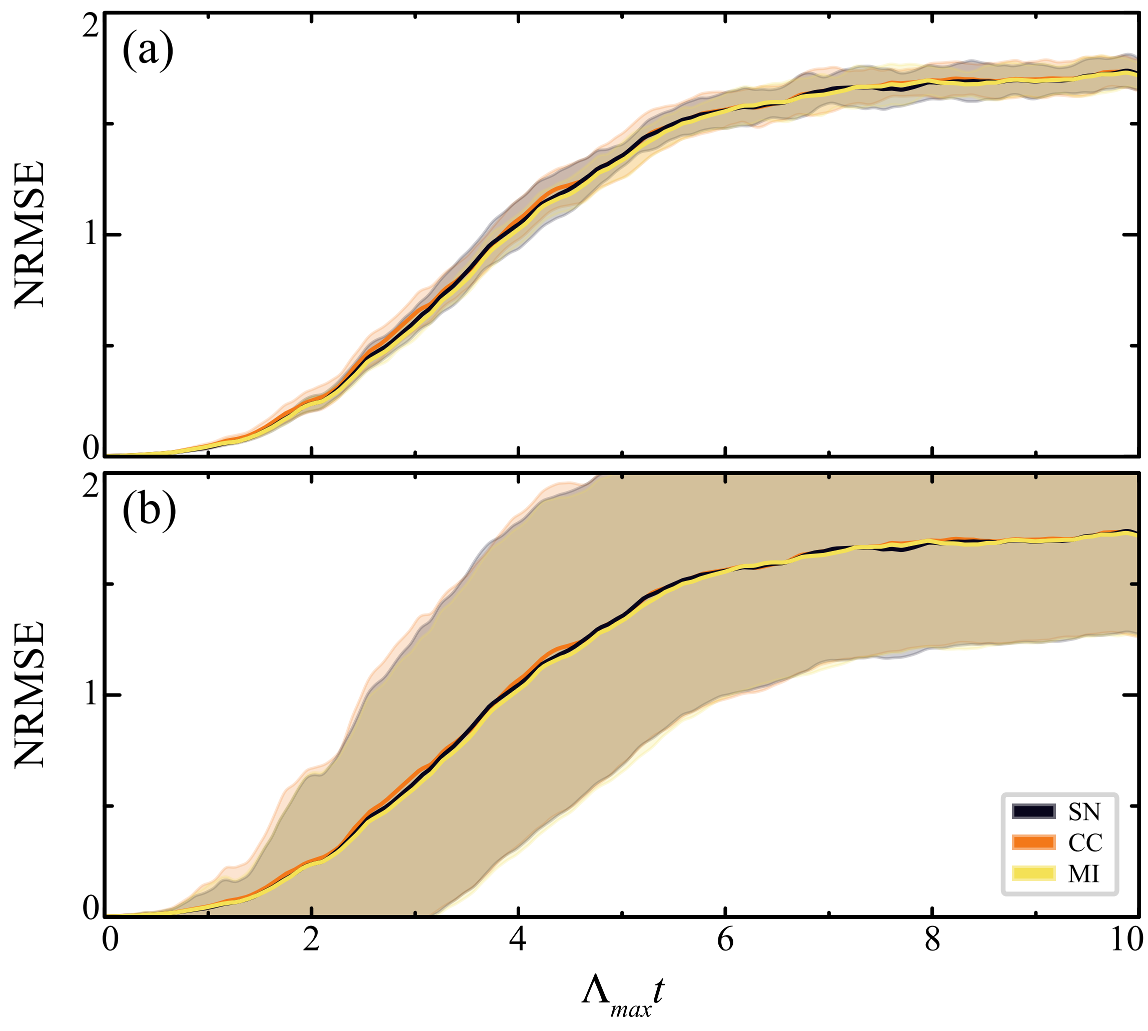

Figure A.2: NRMSE for the KS prediction done in 5.3.2. **(a)** NRMSE averaged first over all 300 different starting positions and then again over all 30 random network realizations. The error bands correspond to the 3σ standard deviation of the random network realizations of only this latter set, the 30 realizations. **(b)** NRMSE averaged over all 9000 independent predictions at once. The error bands correspond to the 1σ standard deviation of these 9000 distinct predictions.

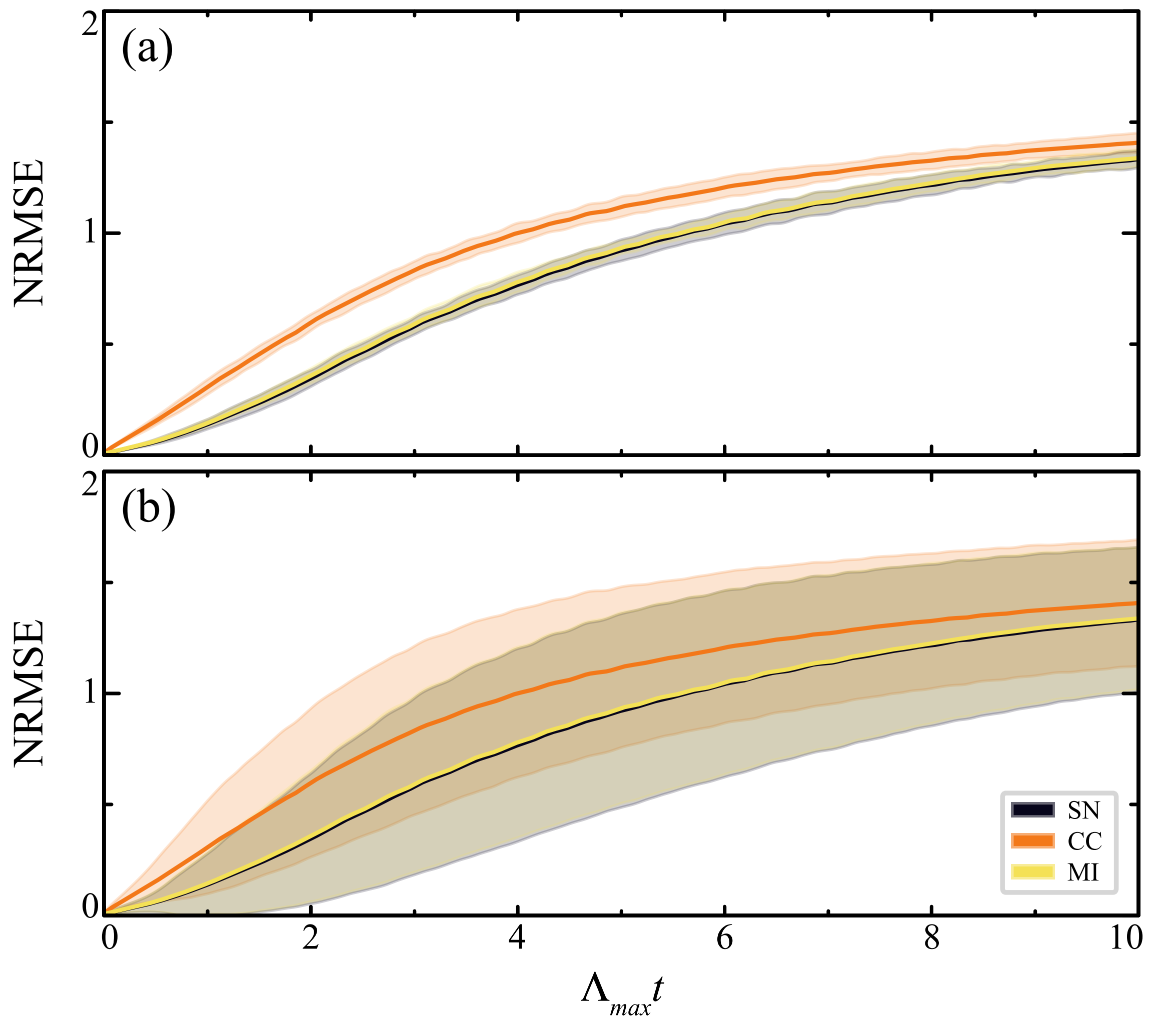

Figure A.3: NRMSE for the L96 prediction done in 5.3.2. **(a)** NRMSE averaged first over all 300 different starting positions and then again over all 30 random network realizations. The error bands correspond to the 3σ standard deviation of the random network realizations of only this latter set, the 30 realizations. **(b)** NRMSE averaged over all 9000 independent predictions at once. The error bands correspond to the 1σ standard deviation of these 9000 distinct predictions.

A.3 Anaconda Environment

In the interest of reproducibility, listed here is the full environment file for the open source package distribution software Anaconda [46]. All code was written in Python 3.7.5 [47]. and executed using the packages specified in the following environment.

```
name: rc_env
channels:
- defaults
dependencies:
- _anaconda_depends=2019.03=py37_0
- _ipyw_jlab_nb_ext_conf=0.1.0=py37_0
- _libgcc_mutex=0.1=main
- alabaster=0.7.12=py37_0
- anaconda=custom=py37_1
- anaconda-client=1.7.2=py37_0
- anaconda-navigator=1.9.7=py37_0
- anaconda-project=0.8.4=py_0
- argh=0.26.2=py37_0
- asn1crypto=1.2.0=py37_0
- astroid=2.3.3=py37_0
- astropy=3.2.3=py37h7b6447c_0
- atomicwrites=1.3.0=py37_1
- attrs=19.3.0=py_0
- autopep8=1.4.4=py_0
- babel=2.7.0=py_0
- backcall=0.1.0=py37_0
- backports=1.0=py_2
- backports.functools_lru_cache=1.6.1=py_0
- backports.os=0.1.1=py37_0
- backports.shutil_get_terminal_size=1.0.0=py37_2
- backports.tempfile=1.0=py_1
- backports.weakref=1.0.post1=py_1
- beautifulsoup4=4.8.1=py37_0
- bitarray=1.1.0=py37h7b6447c_0
- bkcharts=0.2=py37_0
- blas=1.0=mkl - bleach=3.1.0=py37_0
- blosc=1.16.3=hd408876_0
- bokeh=1.4.0=py37_0
- boto=2.49.0=py37_0
- bottleneck=1.3.1=py37hdd07704_0
- bzip2=1.0.8=h7b6447c_0
- ca-certificates=2020.6.24=0
- cairo=1.14.12=h8948797_3
- certifi=2020.6.20=py37_0
- cffi=1.14.0=py37h2e261b9_0
- chardet=3.0.4=py37003
- click=7.0=py37_0
- cloudpickle=1.2.2=py_0
- clyent=1.2.2=py37_1
- colorama=0.4.3=py_0
- conda-package-handling=1.6.0=py37h7b6447c_0
- conda-verify=3.4.2=py_1
- contextlib2=0.6.0.post1=py_0
- cryptography=2.8=py37h1ba5d50_0
- curl=7.67.0=hbc83047_0
- cycler=0.10.0=py37_0
- cython=0.29.15=py37he6710b0_0
- cytoolz=0.10.1=py37h7b6447c_0
- dask=2.9.0=py_0
- dask-core=2.9.0=py_0
- dbus=1.13.12=h746ee38_0
- decorator=4.4.1=py_0
- defusedxml=0.6.0=py_0
- diff-match-patch=20181111=py_0
- distributed=2.9.0=py_0
- docutils=0.15.2=py37_0
- entrypoints=0.3=py37_0
- et_xmlfile=1.0.1=py37_0
- expat=2.2.6=he6710b0_0
- fastcache=1.1.0=py37h7b6447c_0
- filelock=3.0.12=py_0
- flake8=3.7.9=py37_0
- flask=1.1.1=py_0
- fontconfig=2.13.0=h9420a91_0
- freetype=2.9.1=h8a8886c_1
- fribidi=1.0.5=h7b6447c_0
- fsspec=0.6.0=py_0
- future=0.18.2=py37_0
- get_terminal_size=1.0.0=haa9412d_0
- gevent=1.4.0=py37h7b6447c_0
- glib=2.63.1=h5a9c865_0
- glob2=0.7=py_0
- gmp=6.1.2=h6c8ec71_1
- gmpy2=2.0.8=py37h10f8cd9_2
- graphite2=1.3.13=h23475e2_0
- greenlet=0.4.15=py37h7b6447c_0
- gst-plugins-base=1.14.0=hbbd80ab_1
- gstreamer=1.14.0=hb453b48_1
- h5py=2.9.0=py37h7918eee_0
- harfbuzz=1.8.8=hffaf4a1_0
- hdf5=1.10.4=hb1b8bf9_0
- heapdict=1.0.1=py_0
- html5lib=1.0.1=py37_0
- hypothesis=4.44.2=py37_0
- icu=58.2=h9c2bf20_1
- idna=2.8=py37_0
- imageio=2.6.1=py37_0
- imagesize=1.1.0=py37_0
- importlib_metadata=1.3.0=py37_0
- intel-openmp=2019.4=243
- intervaltree=3.0.2=py_0
- ipykernel=5.1.3=py37h39e3cac_1
- ipython=7.10.2=py37h39e3cac_0
- ipython_genutils=0.2.0=py37_0
- ipywidgets=7.5.1=py_0
- isort=4.3.21=py37_0
- itsdangerous=1.1.0=py37_0
- jbig=2.1=hdba287a_0
- jdcal=1.4.1=py_0
- jedi=0.14.1=py37_0
- jeepney=0.4.1=py_0
- jinja2=2.10.3=py_0
- joblib=0.14.1=py_0
- jpeg=9b=h024ee3a_2
- json5=0.8.5=py_0
- jsonschema=3.2.0=py37_0
- jupyter=1.0.0=py37_7
- jupyter_client=5.3.4=py37_0
- jupyter_console=6.0.0=py37_0
- jupyter_core=4.6.1=py37_0
- jupyterlab=1.2.3=pyhf63ae98_0
- jupyterlab_server=1.0.6=py_0
- keyring=19.2.0=py37_0
- kiwisolver=1.1.0=py37he6710b0_0
- krb5=1.16.4=h173b8e3_0
- lazy-object-proxy=1.4.3=py37h7b6447c_0
- libarchive=3.3.3=h5d8350f_5
- libcurl=7.67.0=h20c2e04_0
- libedit=3.1.20181209=hc058e9b_0
- libffi=3.2.1=hd88cf55_4
- libgcc-ng=9.1.0=hdf63c60_0
- libgfortran-ng=7.3.0=hdf63c60_0
- liblief=0.9.0=h7725739_2
- libpng=1.6.37=hbc83047_0
- libsodium=1.0.16=h1bed415_0
- libspatialindex=1.9.3=he6710b0_0
- libssh2=1.8.2=h1ba5d50_0
- libstdcxx-ng=9.1.0=hdf63c60_0
- libtiff=4.1.0=h2733197_0
- libtool=2.4.6=h7b6447c_5
- libuuid=1.0.3=h1bed415_2
- libxcb=1.13=h1bed415_1
- libxml2=2.9.9=hea5a465_1
- libxslt=1.1.33=h7d1a2b0_0
- llvmlite=0.30.0=py37hd408876_0
- locket=0.2.0=py37_1
- lxml=4.4.2=py37hefd8a0e_0
- lz4-c=1.8.1.2=h14c3975_0
- lzo=2.10=h49e0be7_2
- markupsafe=1.1.1=py37h7b6447c_0
- matplotlib=3.1.1=py37h5429711_0
- mccabe=0.6.1=py37_1
- mistune=0.8.4=py37h7b6447c_0
- mkl=2019.4=243
- mkl-service=2.3.0=py37he904b0f_0
- mkl_fft=1.0.15=py37ha843d7b_0
- mkl_random=1.1.0=py37hd6b4f25_0
- mock=3.0.5=py37_0
- more-itertools=7.2.0=py37_0
- mpc=1.1.0=h10f8cd9_1
- mpfr=4.0.1=hdf1c602_3
- mpi=1.0=mpich - mpi4py=3.0.3=py37h028fd6f_0
- mpich=3.3.2=hc856adb_0
- mpmath=1.1.0=py37_0
- msgpack-python=0.6.1=py37hfd86e86_1
- multipledispatch=0.6.0=py37_0
- navigator-updater=0.2.1=py37_0
- nbconvert=5.6.1=py37_0
- nbformat=4.4.0=py37_0
- ncurses=6.1=he6710b0_1
- networkx=2.4=py_0
- nltk=3.4.5=py37_0
- nose=1.3.7=py37_2
- notebook=6.0.2=py37_0
- numba=0.46.0=py37h962f231_0
- numexpr=2.7.0=py37hd81dba3_0
- numpy=1.17.4=py37hc1035e2_0
- numpy-base=1.17.4=py37hde5b4d6_0
- numpydoc=0.9.1=py_0
- olefile=0.46=py37_0
- openpyxl=3.0.2=py_0
- openssl=1.1.1g=h7b6447c_0
- packaging=19.2=py_0
- pandas=0.25.3=py37he6710b0_0
- pandoc=2.2.3.2=0
- pandocfilters=1.4.2=py37_1
- pango=1.42.4=h049681c_0
- parso=0.5.2=py_0
- partd=1.1.0=py_0
- patchelf=0.10=he6710b0_0
- path=13.1.0=py37_0
- path.py=12.4.0=0
- pathlib2=2.3.5=py37_0
- pathtools=0.1.2=py_1
- patsy=0.5.1=py37_0
- pcre=8.43=he6710b0_0
- pep8=1.7.1=py37_0
- pexpect=4.7.0=py37_0
- pickleshare=0.7.5=py37_0
- pillow=6.2.1=py37h34e0f95_0
- pip=19.3.1=py37_0
- pixman=0.38.0=h7b6447c_0
- pkginfo=1.5.0.1=py37_0
- pluggy=0.13.1=py37_0
- ply=3.11=py37_0
- prometheus_client=0.7.1=py_0
- prompt_toolkit=2.0.10=py_0
- psutil=5.6.7=py37h7b6447c_0
- ptyprocess=0.6.0=py37_0
- py=1.8.0=py37_0
- py-lief=0.9.0=py37h7725739_2
- pycodestyle=2.5.0=py37_0
- pycosat=0.6.3=py37h7b6447c_0
- pycparser=2.19=py37_0
- pycrypto=2.6.1=py37h14c3975_9
- pycurl=7.43.0.3=py37h1ba5d50_0
- pydocstyle=4.0.1=py_0
- pyflakes=2.1.1=py37_0
- pygments=2.5.2=py_0
- pylint=2.4.4=py37_0
- pympler=0.7=py_0
- pyodbc=4.0.27=py37he6710b0_0
- pyopenssl=19.1.0=py37_0
- pyparsing=2.4.5=py_0
- pyqt=5.9.2=py37h05f1152_2
- pyrsistent=0.15.6=py37h7b6447c_0
- pysocks=1.7.1=py37_0
- pytables=3.6.1=py37h71ec239_0
- pytest=5.3.2=py37_0
- pytest-arraydiff=0.3=py37h39e3cac_0
- pytest-astropy=0.7.0=py_0
- pytest-astropy-header=0.1.1=py_0
- pytest-doctestplus=0.5.0=py_0
- pytest-openfiles=0.4.0=py_0
- pytest-remotedata=0.3.2=py37_0
- python=3.7.5=h0371630_0
- python-dateutil=2.8.1=py_0
- python-jsonrpc-server=0.3.2=py_0
- python-language-server=0.31.2=py37_0
- python-libarchive-c=2.8=py37_3
- pytz=2019.3=py_0
- pywavelets=1.1.1=py37h7b6447c_0
- pyxdg=0.26=py_0
- pyyaml=5.2=py37h7b6447c_0
- pyzmq=18.1.0=py37he6710b0_0
- qdarkstyle=2.7=py_0
- qt=5.9.7=h5867ecd_1
- qtawesome=0.6.0=py_0
- qtconsole=4.6.0=py_1
- qtpy=1.9.0=py_0
- readline=7.0=h7b6447c_5
- requests=2.22.0=py37_1
- ripgrep=0.10.0=hc07d326_0
- rope=0.14.0=py_0
- rtree=0.8.3=py37_0
- ruamel_yaml=0.15.46=py37h14c3975_0
- scikit-image=0.15.0=py37he6710b0_0
- scikit-learn=0.21.3=py37hd81dba3_0
- scipy=1.3.2=py37h7c811a0_0
- seaborn=0.9.0=pyh91ea838_1
- secretstorage=3.1.1=py37_0
- send2trash=1.5.0=py37_0
- setuptools=42.0.2=py37_0
- simplegeneric=0.8.1=py37_2
- singledispatch=3.4.0.3=py37_0
- sip=4.19.8=py37hf484d3e_0
- six=1.13.0=py37_0
- snappy=1.1.7=hbae5bb6_3
- snowballstemmer=2.0.0=py_0
- sortedcollections=1.1.2=py37_0
- sortedcontainers=2.1.0=py37_0
- soupsieve=1.9.5=py37_0
- sphinx=2.3.0=py_0
- sphinxcontrib=1.0=py37_1
- sphinxcontrib-applehelp=1.0.1=py_0
- sphinxcontrib-devhelp=1.0.1=py_0
- sphinxcontrib-htmlhelp=1.0.2=py_0
- sphinxcontrib-jsmath=1.0.1=py_0
- sphinxcontrib-qthelp=1.0.2=py_0
- sphinxcontrib-serializinghtml=1.1.3=py_0
- sphinxcontrib-websupport=1.1.2=py_0
- spyder=4.0.0=py37_0
- spyder-kernels=1.8.1=py37_0
- sqlalchemy=1.3.11=py37h7b6447c_0
- sqlite=3.30.1=h7b6447c_0
- statsmodels=0.10.1=py37hdd07704_0
- sympy=1.5.1=py37_0
- tbb=2019.8=hfd86e86_0
- tblib=1.5.0=py_0
- terminado=0.8.3=py37_0
- testpath=0.4.4=py_0
- tk=8.6.8=hbc83047_0
- toolz=0.10.0=py_0
- tornado=6.0.3=py37h7b6447c_3
- tqdm=4.40.2=py_0
- traitlets=4.3.3=py37_0
- ujson=1.35=py37h14c3975_0
- unicodecsv=0.14.1=py37_0
- unixodbc=2.3.7=h14c3975_0
- urllib3=1.25.7=py37_0
- watchdog=0.9.0=py37_1
- wcwidth=0.1.7=py37_0
- webencodings=0.5.1=py37_1
- werkzeug=0.16.0=py_0
- wheel=0.33.6=py37_0
- widgetsnbextension=3.5.1=py37_0
- wrapt=1.11.2=py37h7b6447c_0
- wurlitzer=2.0.0=py37_0
- xlrd=1.2.0=py37_0
- xlsxwriter=1.2.6=py_0
- xlwt=1.3.0=py37_0
- xz=5.2.4=h14c3975_4
- yaml=0.1.7=had09818_2
- yapf=0.28.0=py_0
- zeromq=4.3.1=he6710b0_3
- zict=1.0.0=py_0
- zipp=0.6.0=py_0
- zlib=1.2.11=h7b6447c_3
- zstd=1.3.7=h0b5b093_0
```

Acknowledgement

I would like to express my gratitude to all the people who supported me during the my thesis. In particular I would like to thank

Christoph for enabling and supervising this thesis, his positivity and our many discussions making me a better researcher.

Hubertus for facilitating a friendly but productive work environment and allowing me the freedom to work on interesting programming challenges.

Jonas, Joschka and Youssef for our discussions and their help in solving a variety physics and programming problems.

Daniel M. and Peter for going above and beyond when helping me with my computer troubles.

Daniel E., Prapti and Eshita for the company in the office before the coronavirus hit.

Alexander and Haochun for our short but fascinating knowledge exchanges.

All members of the Forschungsgruppe Komplexe Plasmen for the pleasant atmosphere and fun conversations.

Many of my friends for taking hours out of their day to proof read this thesis.

My family for their relentless support of my studies and life in general.

Declaration

I hereby declare that this thesis is my own work, and that I have not used any sources and aids other than those stated in the thesis.

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

Braunschweig, September 15, 2020

Sebastian Baur